基于MATLAB/Simulink的

系统仿真技术与应用

薛定宇　陈阳泉 著

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书首先介绍了MATLAB语言的程序设计的基本内容，在此基础上系统介绍了系统仿真所必要的数值计算方法及MATLAB实现，并以Simulink为主要工具介绍了系统仿真方法与技巧，包括连续系统、离散系统、随机输入系统和复数系统的仿真，由浅入深介绍了模块封装技术、电力系统模块集、非线性系统设计模块集、S-函数编写与应用、Stateflow有限状态机、虚拟现实工具箱等中高级使用方法，最后还介绍了半实物仿真技术与实时控制技术。

本书可作为一般读者学习和掌握MATLAB/Simulink语言的教科书，也可作为高校理工科各类专业的本科生和研究生系统仿真类课程的教材和参考书，还可供科技工作者、教师作为学习和应用系统仿真分析技术解决实际问题的参考资料。

书　　名：基于 MATLAB/Simulink 的系统仿真技术与应用
作　　者：薛定宇　陈阳泉　　著
出 版 者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
　　　　　http://www.tup.tsinghua.edu.cn
责任编辑：欧振旭
印 刷 者：北京昌平环球印刷厂
发 行 者：新华书店总店北京发行所
开　　本：787×1092 1/16　印张：28　字数：645 千字
版　　次：2002 年 4 月第 1 版　2002 年 4 月第 1 次印刷
书　　号：ISBN 7-302-05341-3/TP・3137
印　　数：0001～6000
定　　价：38.00 元

前　言

子曰：**“工欲善其事，必先利其器”**（《论语・卫灵公篇》）。如果有一种十分有效的工具能容易地解决在系统仿真及领域的教学与研究中遇到的问题，它可以将使用者从繁琐、无谓的底层编程中解放出来，把有限的宝贵时间更多地花在解决科学问题中，这样无疑会提高工作效率。本书介绍的 MATLAB®/Simulink® 语言正是这样的一种工具。在系统仿真、自动控制等领域，国外很多高校在教学与研究中都将 MATLAB/Simulink 语言作为首选的计算机工具。我国的科学工作者和教育工作者也逐渐认识到 MATLAB 语言的重要性，对 MATLAB 语言在理工科教学与科研中的地位也达成了共识。

MATLAB 语言虽然是计算数学专家倡导并开发的，但其普及和发展离不开自动控制领域学者的贡献。甚至可以说，MATLAB 语言是自动控制领域学者和工程技术人员捧红的，因为在 MATLAB 语言的发展进程中，许多有代表性的成就和控制界的要求与贡献是分不开的。迄今为止，大多数工具箱也都是控制方面的。MATLAB 具有强大的数学运算能力、方便实用的绘图功能及语言的高度集成性，它在其他科学与工程领域的应用也是越来越广，并且有着更广阔的应用前景和无穷无尽的潜能。

多年来，本人一直在试图以最实用的方式将 MATLAB 语言介绍给国内的读者，并在清华大学出版社出版了 3 部有关 MATLAB 语言及其应用方面的著作，受到了国内外广大中文读者的普遍欢迎。其中，1996 年出版的《控制系统计算机辅助设计——MATLAB 语言与应用》一书被公认为国内关于 MATLAB 语言方面书籍中出版最早、影响最广的著作。本书主要介绍目前最新的 MATLAB 6.1/Simulink 4.1 版本，并适当引入即将正式推出的 MATLAB 6.5/Simulink 5.0 版本的新内容。

作者从 1988 年开始系统地使用 MATLAB 语言进行程序设计与科学研究，积累了丰富的第一手经验；用 MATLAB 语言编写的程序曾作为英国 Rapid Data 软件公司的商品在国际范围内发行，新近编写的几个通用程序在 The MathWorks 公司 (MATLAB 语言的开发者) 的网页上可以下载，得到了国际上很多用户的关注。

本书的合作者陈阳泉博士在系统仿真和 MATLAB/Simulink 语言应用领域有很深的造诣和独到见解，在控制系统的理论研究和工业过程的半实物仿真与实时控制都颇有建树。

本书从使用者的角度出发，并结合作者十多年的实际编程经验和体会，系统地介绍 MATLAB 语言的编程技术及其在科学运算中的应用，书中融合了作者的许多编程经验和第一手材料，内容精心剪裁，相信仍旧会受到读者的欢迎。

应一些读者和朋友的建议，作者建立了“MATLAB 大观园”网站 (`http://matlab.myrice.com`) 和“MATLAB 语言与应用”交流论坛 (`http://matlab.netsh.net`)，一年多来在广大读者和 MATLAB 语言的使用者中有着良好的声誉和影响，解决了大量用户遇到的实际问题，是读者和使用者交流经验、解决问题或答疑的一方大地，被戏称为作者著作的“售后服务站”。

本书由东北大学信息学院徐心和教授主审。本书从酝酿到整个写作过程始终得到徐老师的鼓励和支持。作者的导师，东北大学任兴权教授和英国 Sussex 大学的 Derek P. Atherton 教授也对本书的最终成形提供了很多的帮助，是他们将作者引入系统仿真和 MATLAB/Simulink 语言编程的乐园，并在这个领域开始了充满趣味的工作。

作者的一些同行和朋友也先后给予作者许多建议和支持，包括北方交通大学机电学院的朱衡君教授、中科院系统科学研究所的韩京清研究员、澳大利亚军事科学院大学学院的谢力博士、东北大学信息学院的王贞祥教授、东北大学材料冶金学院钟圣俊博士、哈尔滨工业大学土木工程学院王刚博士等，还有在互联网上交流的众多知名的和不知名的同行，在此表示深深的谢意。在本书部分内容的写作过程中，作者和博士生胡英、潘峰、龙德、司秉玉、吴炜等同学进行了较深入的讨论，在此一并表示感谢。

本书的出版得到了清华大学出版社蔡鸿程总编的关怀和帮助，还得到出版社的欧振旭编辑细心的加工，作者对他们的辛勤工作深表谢意。

本书受“教育部高等学校骨干教师资助计划”资助，在此深表谢意。

本书的出版还得到了美国 The MathWorks 公司图书计划的支持，在此表示谢意，并特别感谢 Noami Fernandez 女士为作者作出的各种帮助。国外读者如果想购买 MATLAB 软件及系列产品，请和该公司直接联系：

The MathWorks, Inc., 3 Apple Hill Drive
Natick, MA, 01760-2098 USA
电话：+01-508-647-7000，传真：+01-508-647-7101
E-mail：info@mathworks.com
网站：http://www.mathworks.com

国内读者可以直接和该公司在中国的惟一代理九州恒润科技有限公司联系：
北京市北三环中路 27 号商房大厦 620 室，邮编 100029
电话：010-62073595，传真：010-62073600
网站：http://www.hirain.com

出于作者水平有限，书中的缺点错误在所难免，欢迎读者批评指教。

多年来，我的妻子杨军在生活和事业上给予了我莫大的帮助与鼓励，没有她的鼓励和一如既往的支持，本书和前几部著作均不能顺利面世，谨以此书献给她和女儿薛杨。

薛定宇 (xue_dy@21cn.com)
2002 年 1 月 30 日于沈阳东北大学

目 录

第 1 章 系统仿真技术与应用

1.1 系统仿真技术概述

系统是由客观世界中实体与实体间的相互作用和相互依赖关系构成的具有某种特定功能的有机整体。系统的分类方法是多种多样的，习惯上依照其应用范围可以将系统分为工程系统和非工程系统。工程系统的含意是指由相互关联部件组成的一个整体，实现特定的目标，例如电机驱动自动控制系统是由执行部件、功率转换部件、检测部件所组成，用它来完成电机的转速、位置和其他参数控制的某个特定目标。

非工程系统涵盖的范围更加广泛，大至宇宙，小至微观世界都存在着相互关联、相互制约的关系，形成一个整体，实现某种目的，所以均可以认为是系统。

如果想定量地研究系统的行为，可以将其本身的特性及内部的相互关系抽象出来，构造出系统的模型。系统的模型分为物理模型和数学模型。由于计算机技术的迅速发展和广泛应用，数学模型的应用越来越普遍。

系统的数学模型是描述系统动态特性的数学表达式，用来表示系统运动过程中各个量的关系，是分析、设计系统的依据。从它所描述系统的运动性质和数学工具来分，又可以分为连续系统、离散时间系统、离散事件系统、混杂系统等。还可以细分为线性、非线性、定常、时变、集中参数、分布参数、确定性、随机等子类。

系统仿真是根据被研究的真实系统的数学模型研究系统性能的一门学科，现在尤指利用计算机去研究数学模型行为的方法。计算机仿真的基本内容包括系统、模型、算法、计算机程序设计与仿真结果显示、分析与验证等环节。

在系统仿真技术的诸多环节中，算法和计算机程序设计是很重要的一个环节，它直接决定原来问题是否能够正确地求解。基于国际上仿真领域最权威、最实用的计算机工具 —— MATLAB® 语言介绍仿真问题的编程与求解方法将是本书最显著的特点。在第 1.2 节中将介绍数学软件、仿真软件的仿真概况和现状，第 1.3 节着重介绍 MATLAB/Simulink® 语言的发展状况、语言特色和在系统仿真领域的应用举例，可以初步领略 MATLAB 的强大功能，第 1.4 节中将介绍本书的结构和有关内容。

1.2 仿真软件的发展状况与应用

早期的计算机仿真技术大致经历了几个阶段：20 世纪 40 年代模拟计算机仿真；50 年代初数字仿真；60 年代早期仿真语言的出现等。80 年代出现的面向对象仿真技术为系统仿真方法注入了活力。我国早在 50 年代就开始研究仿真技术了，当时主要用于国防领域，以模拟计算机的仿真为主。70 年代初开始应用数字计算机进行仿真[41]。随着数字

计算机的普及，近 20 年以来，国际、国内出现了许多专门用于计算机数字仿真的仿真语言与工具，如 CSMP, ACSL, SIMNON, MATLAB/Simulink, MatrixX/System Build, CSMP-C 等。

1.2.1　早期数学软件包的发展概况

数字计算机的出现给数值计算技术的研究注入了新的活力。在现代计算技术的早期发展中，出现了一些著名的数学软件包，如美国的基于特征值的软件包 EISPACK[13, 44] 和线性代数软件包 LINPACK[8]，英国牛津数值算法研究组 (Numerical Algorithm Group) 开发的 NAG 软件包[38] 及文献 [40] 中给出的程序集等，这些都是在国际上广泛流行的、有着较高声望的软件包。

美国的 EISPACK 和 LINPACK 都是基于矩阵特征值和奇异值解决线性代数问题的专用软件包。限于当时的计算机发展状况，这些软件包大都是由 Fortran 语言编写的源程序组成的。

例如若想求出 N 阶实矩阵 $\boldsymbol{A}$ 的全部特征值 (用 WR, WI 数组分别表示其实虚部) 和对应的特征向量矩阵 Z，则 EISPACK 软件包给出的子程序建议调用路径为：

```
CALL BALANC(NM,N,A,IS1,IS2,FV1)
CALL ELMHES(NM,N,IS1,IS2,A,IV1)
CALL ELTRAN(NM,N,IS1,IS2,A,IV1,Z)
CALL HQR2(NM,N,IS1,IS2,A,WR,WI,Z,IERR)
IF (IERR.EQ.0) GOTO 99999
CALL BALBAK(NM,N,IS1,IS2,FV1,N,Z)
```

由上面的叙述可以看出，要求取矩阵的特征值和特征向量，首先要给一些数组和变量依据 EISPACK 的格式作出定义和赋值，并编写出主程序，再经过编译和连接过程，形成可执行文件，最后才能得出所需的结果。

英国的 NAG 和美国学者的 Numerical Recipes 工具包则包括了各种各样数学问题的数值解法，二者中 NAG 的功能尤其强大。NAG 的子程序都是以字母加数字编号的形式命名的，非专业人员很难找到适合自己问题的子程序，更不用说能保证以正确的格式去调用这些子程序了。这些程序包使用起来极其复杂，谁也不能保证不发生错误，NAG 光数百页的使用手册就十几本！

Numerical Recipes 一书中给出的一系列算法语言源程序也是一个在国际上广泛应用的软件包。该书中的子程序有 C, Fortran 和 Pascal 等版本，适合于科学研究者和工程技术人员直接应用。该书的程序包由 200 多个高效、实用的子程序构成，这些子程序一般有较好的数值特性，比较可靠，为各国的研究者信赖。

具有 Fortran 和 C 等高级计算机语言知识的读者可能已经注意到，如果用它们去进行程序设计，尤其当涉及矩阵运算或画图时，则编程会很麻烦。比如说，若想求解一个线性代数方程，用户得首先去编写一个主程序，然后编写一个子程序去读入各个矩阵的元素，之后再编写一个子程序，求解相应的方程 (如使用 Gauss 消去法)，最后输出计算

结果。如果选择的计算子程序不是很可靠，则所得的计算结果往往可能会出现问题。如果没有标准的子程序可以调用，则用户往往要将自己编好的子程序逐条地输入计算机，然后进行调试，最后进行计算。这样一个简单的问题往往需要用户编写 100 条左右的源程序，输入与调试程序也是很费事的，并无法保证所输入的程序 100% 可靠。求解线性方程组这样一个简单的功能需要 100 条源程序，其他复杂的功能往往要求有更多条语句，如采用双步 QR 法求取矩阵特征值的子程序则需要 500 多条源程序，其中任何一条语句有毛病，甚至调用不当 (如数组维数不匹配) 都可能导致错误结果的出现。

用软件包的形式编写程序有如下的缺点：

- **使用不方便** 对不是很熟悉所使用软件包的用户来说，直接利用软件包编写程序是相当困难的，也是容易出错的。如果其中一个子程序调用发生微小的错误则可能导致最终得出错误的结果。
- **调用过程繁琐** 首先需要编写主程序，确定对软件包的调用过程，再经过必要的编译和连接过程，有时还要花大量的时间去调试程序以保证其正确性，而不是想得出什么马上就可以得出的。
- **执行程序过多** 想求解一个特定的问题就需要编写一个专门的程序，并形成一个可执行文件，如果需要求解的问题很多，那么就需要在计算机硬盘上同时保留很多这样的可执行文件，这样，计算机磁盘空间的利用不是很经济，管理起来也将十分困难。
- **不利于传递数据** 通过软件包调用方式会针对每个具体问题形成一个孤立的可执行文件，因而在一个程序中产生的数据无法传入另一个程序，更无法使几个程序同时执行以解决所关心的问题。
- **维数指定困难** 在很多数学问题中最重要的变量是矩阵，如果要求解的问题维数较低，则形成的程序就不能用于求解高阶问题，例如文献 [33] 中的程序维数均定为 10 阶。所以有时为使得程序通用，往往将维数设置得很大，这样在解小规模问题时会出现空间的浪费，而更大规模问题仍然求解不了。在优秀的软件中往往需要动态地进行矩阵定维。

此外，这里介绍的大多数早期软件包都是由 Fortran 语言编写的，由于众所周知的原因，以前使用 Fortran 语言绘图并不是轻而易举的事情，它需要调用相应的软件包做进一步处理，在绘图方面比较实用和流行的软件包是 GINO-F[5]，但这种软件包只给出绘图的基本子程序，如果要绘制较满意的图形则需要用户自己用这些低级命令编写出合适的绘图子程序来。

除了上面指出的缺点以外，用 Fortran 和 C 等程序设计语言编程还有一个致命的弱点，那就是因为 C 语言本身的原因，致使在不同的机器平台上，扩展的 C 源程序代码是不兼容的，尤其在绘图及界面设计方面更是如此。例如在 PC 机的 Microsoft Windows 操作系统下编写的 C 语言程序不能立即在 SUN 工作站上直接运行，而需要在该机器上对源程序进行修改、编译后才可以执行。

尽管如此，数学软件包仍在继续发展，其发展方向是采用国际上最先进的数值算法，提供更高效的、更稳定的、更快速、更可靠的数学软件包。例如在线性代数计算领

域，全新的 LAPACK 已经成为当前最有影响的软件包，但它们的目的似乎已经不再为一般用户提供解决问题的方法，而是为数学软件提供底层的支持。在新版的 MATLAB 已经抛弃了一直使用的 LINPACK 和 EISPACK，采用 LAPACK 为其底层支持软件包。

1.2.2 仿真软件的发展概况

从前面提及的软件包的局限性看，直接调用它们进行系统仿真将有较大的困难，因为要掌握这些函数的接口是一件相当复杂的事，准确调用它们将更难；此外，软件包函数调用直接得出的结果可信度也不是很高，因为软件包的质量参差不齐。

抛弃成型的软件包另起炉灶自己编写程序也不是很现实的事，毕竟在成型软件包中包含有很多同行专家的心血，有时自己从头编写程序很难达到这样的效果，所以必须采用经验证的信誉著称的高水平软件包或计算机语言来进行仿真研究。

仿真技术引起该领域各国学者、专家们的重视，建立起国际的仿真委员会(Simulation Councils Inc，简称为 SCi)，该公司于 1967 年通过了仿真语言规范。仿真语言 CSMP (Computer Simulation Modelling Language) 应该属于建立在该标准上的最早的专用仿真语言。中科院沈阳自动化研究所在 1988 年推出了该语言的推广版本 —— CSMP-C。

20 世纪 80 年代初期，美国 Mitchell and Gauthier Associate 公司推出了依照该标准的著名仿真语言 ACSL (Advanced Continuous Simulation Language)[34]，该语言出现后，由于其功能较强大，并有一些系统分析的功能，很快就在仿真领域占据了主导地位。

ACSL 首先要求用户依照其语言规则建立起一个模型文件，然后可以通过 ACSL 本身提供的命令对之进行仿真及辅助分析。ACSL 与 FORTRAN 语言的主要区别在于，ACSL 的语句更简练，内容更丰富，ACSL 语言也可以直接调用由 FORTRAN 编写的子程序。ACSL 编程的结构比相应的 FORTRAN 语言更严格。程序的基本结构必须严格按照规定的格式来编写，否则所得出的仿真结果可能出现意想不到的错误。ACSL 提供了几十个系统子模块 (macros)，其中包括很常用的线性和非线性子模块，如传递函数模块 `TRAN`，积分器模块 `INTEG`，超前滞后环节 `LEDLAG`，延迟模块 `DELAY`，死区非线性模块 `DEAD`，磁滞回环 `BAKLSH`，限幅积分器 `LIMINT` 等，用户可以利用这些子模块简单地编写出描述给定系统的仿真模型，然后采用 ACSL 提供的功能来对系统进行仿真分析，并绘制出结果的曲线表示。

编写完 ACSL 源程序后，则可以采用 ACSL 的编译命令来编译并将此模型和 ACSL 库连接起来，则可以形成一个可执行文件。这一过程完成之后 ACSL 将自动给出提示符 `ACSL>`，在这个提示符下用户可以输入相应命令即可。

【例 1.1】 著名的 Van der Pol 方程由下式给出：

$$\ddot{y} + \mu(y^2 - 1)\dot{y} + y = 0$$

若取 $\mu = 1$，并取状态变量为 $y_1 = y, y_2 = \dot{y}$，则 Van der Pol 方程可以写成

$$\dot{y}_1 = y_1(1 - y_2^2) - y_2,\ \dot{y}_2 = y_1$$

这时可以由 ACSL 所定义的语言写出此系统的模型如下:

```
PROGRAM VAN DER POL EQUATION
CINTERVAL CINT=0.01
CONSTANT  Y1C=3.0, Y2C=2.5, TSTP=15.0
   Y1=INTEG(Y1*(1-Y2**2)-Y2, Y1C)
   Y2=INTEG(Y1, Y2C)
TERMT (T.GE.TSTP)
END
```

在此程序中把显示步长 CINT 设置为 0.01，状态变量的初值由 Y1C 和 Y2C 表示，而终止仿真时间 TSTP 设置为 15，该程序中变量 T 为实际仿真时间。这样需要输入如下的语句:

```
ACSL> PREPAR T, Y1, Y2
ACSL> START
ACSL> PLOT Y1, Y2
```

来通知 ACSL 模型在仿真时需要保留 T, Y1, Y2 三个参数，开始数字仿真，并绘制出系统的相平面图 (Y1 和 Y2 直接的关系曲线)。还可以用下面的命令修改系统内部的参数:

```
ACSL> SET Y1C=-1, Y2C=-3
```

但必须先知道系统内部的变量名。

和 ACSL 大致同时的还有瑞典 Lund 工学院 Karl Åström 教授主持开发的 SIMNON，英国 Salford 大学的 ESL 等，这些语言的编程语句结构也是很类似的，因为它们所依据的标准都是相同的。

MATLAB 语言的出现将数值计算技术与应用带入了一个新的阶段，与之配套的 Simulink 仿真环境又为系统仿真技术提供了新的解决方案。MATLAB 语言发行后国际上又出现了很多仿照其思想的软件，如稍后出现的美国的商品软件 Ctrl-C, Matrix-X, O-Matrix，韩国汉城国立大学权旭铉教授主持开发的 CemTool, 以及现在仍作为免费软件的 Octave, SciLAB 等，本书将以当前最新的 MATLAB 6.1/Simulink 4.1 版本为主要对象，系统地介绍 MATLAB 语言的编程技术及其在系统仿真领域的应用。

计算机代数系统是在本领域中又一个吸引人的主题，而解决数学问题解析计算又是 C 类语言直接应用的难点。于是国际上很多学者在研究、开发高质量的计算机代数系统。早期 IBM 公司开发的 mumath 和 Reduce 等软件为解决这样的问题提出了新的思路。后来出现的 Maple 和 Mathematica 逐渐占领了计算机代数系统的市场，成为比较成功的实用工具。

早期的 Mathematica 可以和 MATLAB 语言交互信息，比如通过一个称为 MathLink 的软件接口就可以很容易地完成这样的任务。为了解决计算机代数问题，MATLAB 语言的开发者 —— 美国 The MathWorks 公司也研制开发了“符号运算工具箱” (Symbolic Toolbox®)，该工具箱将 Maple 语言的内核作为 MATLAB 符号运算的引擎，使得二者能更好地结合起来。

这些软件和语言还是很昂贵的，所以有人更倾向于采用免费的，但编程结构类似于

MATLAB 的计算机语言，如 Octave 和 SciLAB，这些软件的全部源程序也是公开的，有较高的透明度，但目前它们的功能已经无法与越来越强大的 MATLAB 语言相比。

子曰："工欲善其事，必先利其器"。系统仿真领域有很多自己的特性，如果能选择一种能反映当今系统仿真领域最高水平，也是最实用的软件或语言介绍仿真技术，使得读者能直接采用该语言解决自己的问题，将是很有意义的。实践证明，MATLAB 就是这样的仿真软件，由于它本身卓越的功能，已经使得它成为自动控制、航空航天、汽车设计等诸多领域仿真的首选语言。所以在本书中将介绍基于 MATLAB/Simulink 的系统仿真理论与应用。

1.3 MATLAB 语言简介

1.3.1 MATLAB 语言发展简史

MATLAB 语言的首创者 Cleve Moler 教授在数值分析，特别是在数值线性代数的领域中很有影响[8, 9, 10, 13, 19, 44]。他曾在密西根大学、斯坦福大学和新墨西哥大学任数学与计算机科学教授。1980 年前后，时任新墨西哥大学计算机系主任的 Moler 教授在讲授线性代数课程时，发现了用其他高级语言编程极为不便，便构思并开发了 MATLAB (MATrix LABoratory，即矩阵实验室)，这一软件利用了他研制的、在国际上颇有影响的 EISPACK[44] (基于特征值计算的软件包) 和 LINPACK[8] (线性代数软件包) 两大软件包中可靠的子程序，用 Fortran 语言编写了集命令翻译、科学计算于一身的一套交互式软件系统。

所谓交互式语言，是指用户给出一条命令，立即就可以得出该命令的结果。该语言无需像 C 和 Fortran 语言那样，首先要求使用者去编写源程序，然后对之进行编译、连接，最终形成可执行文件。这无疑会给使用者带来极大的方便。在 MATLAB 下，矩阵的运算变得异常的容易，所以它一出现就广受欢迎，这一系统逐渐发展、完善，逐步走向成熟，形成了今天的模样。

早期的 MATLAB 只能作矩阵运算；绘图也只能用极其原始的方法，即用星号描点的形式画图；内部函数也只提供了几十个。但即使其当时的功能十分简单，当它作为免费软件出现以来，还是吸引了大批的使用者。

Cleve Moler 和 John Little 等人成立了一个名叫 The MathWorks 的公司，Cleve Moler 一直任该公司的首席科学家。该公司于 1984 年推出了第一个 MATLAB 的商业版本。当时的 MATLAB 版本已经用 C 语言作了完全的改写，其后又增添了丰富多彩的图形图像处理、多媒体功能、符号运算和它与其他流行软件的接口功能，使得 MATLAB 的功能越来越强大。最早的 PC 机版又称为 PC-MATLAB，其工作站版本又称为 Pro MATLAB。1990 年推出的 MATLAB 3.5i 版是第一个可以运行于 Microsoft Windows 下的版本，它可以在两个窗口上分别显示命令行计算结果和图形结果。稍后推出的 SimuLAB 环境首次引入了基于框图的仿真功能，其模型输入的方式令人耳目一新，该环境就是我们现在所知的 Simulink。

The MathWorks 公司于 1992 年推出了具有划时代意义的 MATLAB 4.0 版本，并于 1993 年推出了其微机版，充分支持在 Microsoft Windows 进行界面编程。1994 年推出的 4.2 版本扩充了 4.0 版本的功能，尤其在图形界面设计方面更提供了新的方法。

1997 年推出的 MATLAB 5.0 版支持了更多的数据结构，如单元数据、数据结构体、多维数组、对象与类等，使其成为一种更方便、完美的编程语言。1999 年初推出的 MATLAB 5.3 版在很多方面又进一步改进了 MATLAB 语言的功能，随之推出的全新版本的最优化工具箱和 Simulink 3.0 版达到了很高的档次。2000 年 10 月，MATLAB 6.0 问世，在操作界面上有了很大改观，同时还给出了程序发布窗口、历史信息窗口和变量管理窗口等，为用户的使用提供了很大的方便；在计算内核上抛弃了其一直使用的 LINPACK 和 EISPACK，而采用了更具优势的 LAPACK 软件包和 FFTW 系统，速度变得更快，数值性能也更好；在用户图形界面设计上也更趋合理；与 C 语言接口及转换的兼容性也更强；与之配套的 Simulink 4.0 版的新功能也特别引人注目。2001 年 6 月推出的 MATLAB 6.1 版及 Simulink 4.1 版是目前的最新版本，功能已经十分强大，其新的虚拟现实工具箱更给仿真结果三维视景下显示带来了新的解决方案。

The MathWorks 公司目前正在致力于新版本的开发、测试，将于 2002 年 6 月正式推出 MATLAB Release 13, 即 MATLAB 6.5/Simulink 5.0。从测试版看，似乎新版本与现在的 MATLAB 6.1/Simulink 4.1 相差不是很大，所以本书以 MATLAB 6.1 为主介绍 MATLAB/Simulink 及其应用，并适当介绍 Release 13 中的新特色。

目前，MATLAB 已经成为国际上最流行的科学与工程计算的软件工具，现在的 MATLAB 已经不仅仅是一个“矩阵实验室”了，它已经成为了一种具有广泛应用前景的、全新的计算机高级编程语言了，有人称它为“第四代”计算机语言，它在国内外高校和研究部门正扮演着重要的角色。MATLAB 语言的功能也越来越强大，不断适应新的要求提出新的解决方法。另外，很多长期以来对 MATLAB 有一定竞争能力的软件 (如 Matrix-X) 已经被 The MathWorks 公司吞并，所以可以预见，在科学运算与系统仿真领域 MATLAB 语言将长期保持其独一无二的地位。

1.3.2 MATLAB 语言的特色

除了 MATLAB 语言的强大数值计算和图形功能外，它还有其他语言难以比拟的功能，此外，它和其他语言的接口能够保证它可以和各种各样的强大计算机软件相结合，发挥更大的作用。

MATLAB 目前可以在各种类型的常用计算机上运行，如在 PC 兼容微型机、Sun Sparc 工作站、Silicon Graphics 工作站、惠普工作站、Solaris 工作站、Dec Alpha 工作站、VAX/VMS 系统机、苹果 Macintosh 微型机和其他一些机器上完全兼容。如果单纯地使用 MATLAB 语言进行编程而不采用其他外部语言，则用 MATLAB 语言编写出来的程序不做丝毫的修改便可以直接移植到其他机型上使用，所以说与其他语言不同，MATLAB 是和机器类型和操作系统基本上无关的。

依作者的观点，MATLAB 和其他高级语言之间的关系仿佛该高级语言和汇编语言的

关系一样，因为虽然高级语言的执行效率要低于汇编语言，但其编程效率与可读性、可移植性要远远高于汇编语言。同样，MATLAB 比一般高级语言的执行效率要低，而其编程效率与可读性、可移植性要远远高于其他高级语言，所以在科学运算中较适合于从像 MATLAB 这样的专用高级语言入手，这不但可以大大地提高编程的效率，而且可以大大地提高编程的质量与可靠性。例如在实际应用中，往往需要求出一个矩阵的特征值，而求取矩阵特征值的方法是多种多样的，如 QR 双步方法、Jacobi 方法等，所得出的结果也不尽相同。但不管矩阵的维数和内容如何，在 MATLAB 环境下用一条指令就可以立即求出系统的特征值来，而不必去考虑是用什么算法以及如何实现等低级问题，也不必深入了解相应算法的具体内容。这就像在 C 语言下不必去深究乘法是怎样实现的一样，而只采用乘积的结果就可以了。其实对于专门从事科学运算与系统仿真研究的人员来说，因为 MATLAB 语言可以轻易地再现 C 或 Fortran 语言几乎全部的功能，所以即使用户不懂 C 或 Fortran 这样的程序设计语言也照样可以设计出功能强大、界面优美、稳定可靠的高质量程序来，且开发周期会大大地缩短，可靠性与可信度大大提高。

MATLAB 语言具有较高的运算精度。一般情况下，在矩阵类运算中往往可达到 10^{-15} 数量级的精度，这当然符合一般科学与工程运算的要求。

如果矩阵的条件数很大，则矩阵中一个参数的微小变化，就可能会使最终结果发生极大的变化，这种现象在数学上称为坏条件问题。如果采用的算法不当，则最后得出的结果可能是不正确的。采用 MATLAB 一般不会出现这类错误结果，亦即 MATLAB 是可靠的、数值稳定的，而采用 C 或其他高级语言编写出来的程序在求解这类问题时，如选择的算法不当则可能得出错误结果。

MATLAB 是以复数矩阵作为基本编程单元的一种高级程序设计语言，它提供了各种矩阵的运算与操作，并有较强的绘图功能，所以得以广为流传，成为当今国际科学与工程领域中应用最广、最受人们喜爱的一种软件环境。MATLAB 是一个高度集成的软件系统，它集科学与工程计算、图形可视化、图像处理、多媒体处理于一身，并提供了实用的 Windows 图形界面设计方法，使用户能设计出友好的图形界面。MATLAB 语言在自动控制、航天工业、汽车工业、生物医学工程、语音处理、图像信号处理、雷达工程、信号分析、计算机技术等各行各业中都有极广泛的应用。

1.3.3 MATLAB/Simulink 在仿真中的应用演示

本节中将用几个例子来展示 MATLAB/Simulink 在系统仿真中的应用，这里侧重于给出例子的概要和特色，更详细的内容与实际实现将在后面章节中详细介绍，所以这里并不要求没有 MATLAB 编程经验的读者能真正读懂所涉及的语句。希望通过这里的简单例子，读者能领略 MATLAB/Simulink 的功能与特色，为本书内容的学习和掌握建立起一定的感性认识。另外，MATLAB 的演示程序 demo 将给用户提供各种各样的功能演示，运行此程序会对读者了解 MATLAB/Simulink 有很大的帮助。

【例 1.2】 **线性代数问题的计算机求解** —— *考虑金庸作品中经常提及的一个“数学问题”，该问题用数学语言描述就是：如何生成一个 3×3 矩阵，并将自然数 1, 2, ···, 9 分别置成这 9 个矩阵元素，*

才能使得每一行、每一列、且主、反对角线上元素相加都等于一个相同的数。

MATLAB 语言提供了一个函数 magic() 来生成这样的魔方矩阵，只需在 MATLAB 环境的提示符[①] 下键入 A=magic(3) 命令，就可以立即得出这样的一个矩阵，将此矩阵赋给变量 A，并存入 MATLAB 工作空间。

```
>> A=magic(3)  % 构造 3x3 魔方矩阵，并赋给变量 A
A =
     8     1     6
     3     5     7
     4     9     2
```

其中百分号 (%) 引导的部分为注释，在程序执行时不起任何作用。我们可以手动检查该矩阵，可以发现，这个共同的常数为 15.

诚然，如果让一个小说家去求出更高阶的矩阵实在是勉为其难，然而用 MATLAB 这样强有力的工具，可以立即得出 9×9 这样的高阶魔方矩阵，并可以检验出这个共同的常数为 369。

```
>> B=magic(9) % 生成 9x9 魔方矩阵，赋给变量 B
B =
    47    58    69    80     1    12    23    34    45
    57    68    79     9    11    22    33    44    46
    67    78     8    10    21    32    43    54    56
    77     7    18    20    31    42    53    55    66
     6    17    19    30    41    52    63    65    76
    16    27    29    40    51    62    64    75     5
    26    28    39    50    61    72    74     4    15
    36    38    49    60    71    73     3    14    25
    37    48    59    70    81     2    13    24    35
```

有了矩阵，则可以用 MATLAB 提供的数值线性代数函数进行分析，例如，该矩阵的行列式可以由下面的命令直接解出。

```
>> det(sym(B)) % 行列式解析解，数值解由 det(B) 即可解出
ans =
     75035738059027200
```

另外，inv(B) 可以立即获得 B 矩阵的逆矩阵，用 eig(B) 可以立即得出其特征值，如果用 poly(B) 函数即可获得原矩阵的特征多项式。

〖例 1.3〗 **最优化问题的计算机求解** —— 最优化问题也是在仿真中经常要解决的问题，假设已知最优化问题的数学描述为

$$\min_{\text{s.t.}\begin{cases} x_1^2+x_2^2+x_3^2-25=0 \\ 8x_1+14x_2+7x_3-56=0 \\ x_1,x_2,x_3 \geqslant 0 \end{cases}} \left[1000-x_1^2-2x_2^2-x_3^2-x_1x_2-x_1x_3\right]$$

[①] 其中 >> 为 MATLAB 的提示符，早期的 MATLAB 版本在中文版 Windows 下不能显示此提示符，从 MATLAB 6.0 开始使用正常 ASCII 码表示该提示符，所以将正常显示该提示符。

该问题属于非线性规划问题，可以用下面的语句直接求解出该最优化问题的解:

```
%文件名 examp0
OPT=optimset; OPT.LargeScale='off'; x0=[1;1;1];
LB=[0;0;0]; UB=[]; A=[]; B=[]; Ae=[]; Be=[]; %参数设定
[x,f_opt,c,d]=fmincon(@opt_fun1,x0,A,B,Ae,Be,LB,UB,@opt_con1,OPT);
function y=opt_fun1(x) % 目标函数
y=1000-x(1)*x(1)-2*x(2)*x(2)-x(3)*x(3)-x(1)*x(2)-x(1)*x(3);
function [c,ceq]=opt_con1(x) % 约束函数
ceq=[x(1)*x(1)+x(2)*x(2)+x(3)*x(3)-25; % 等式约束共有两条等式约束
    8*x(1)+14*x(2)+7*x(3)-56];
c = []; % 不等式约束，因为不存在不等式约束，故它为空
```

对照相关的程序段和原始数学模型不难发现其共性。我们将在第3.6节中系统介绍最优化问题的数值解法。运行这个函数就可以立即得出如下的解:

```
>> examp0 % 运行前面编写的 MATLAB 程序
   format long, x   % 以高精度显示问题的解
x =
   3.51212990129051
   0.21698726105162
   3.55216273356475
```

〖例1.4〗 **常微分方程的解** —— 求解常微分方程是一般连续系统仿真的基础。Lorenz方程是一个著名的混沌问题，其数学描述如下:

$$\begin{cases}\dot{x}_1(t)=-8x_1(t)/3+x_2(t)x_3(t)\\ \dot{x}_2(t)=-10x_2(t)+10x_3(t)\\ \dot{x}_3(t)=-x_1(t)x_2(t)+28x_2(t)-x_3(t)\end{cases}$$

若令其初值为 $x_1(0)=x_2(0)=0$, $x_3(0)=10^{-10}$，则可以编写出下面的文件，用它可以求取Lorenz方程的数值解，并绘制解的三维表示:

```
function examp1
[t,y]=ode45(@lorenzeq,[0,100],[0;0;1e-10]); % 求解方程
plot3(y(:,1), y(:,2), y(:,3))  % 绘制三维相轨迹
axis([10 42 -20 20 -20 28]);   % 手动设置坐标系位置
function xdot = lorenzeq(t,x)  % 下面的函数直接对应于给出的微分方程
xdot=[-8/3*x(1)+x(2)*x(3);     % 这三行对应于微分方程的三个方程
      -10*x(2)+10*x(3);
      -x(1)*x(2)+28*x(2)-x(3)];
```

在MATLAB命令窗口中键入 examp1，则可以立即得出Lorenz方程的解，如图1-1所示。

〖例1.5〗 **分形系统仿真** —— 分形系统往往能得出各种美妙的图形，Mandelbrot图是一中最早开发出来的图形，在 $z_{n+1}=z_n^2+c$ 复数映射中，如果 z_0 遍取区域内 $(x_{\min},y_{\min})$ 到 $(x_{\max},y_{\max})$ 所

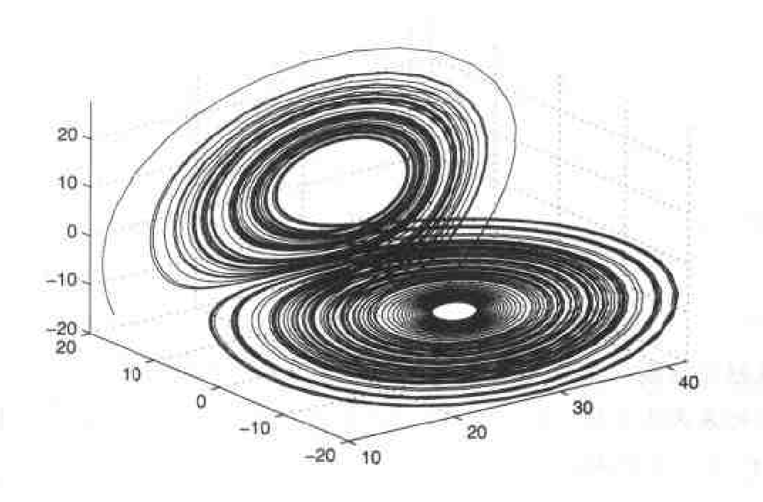


图 1-1 Lorenz 方程的相轨迹曲线

有的点，则可以得出各个点处的测度，从而绘制出 Mandelbrot 图。选定不同的 z_0 值变化区域，则可以绘制出各种各样美妙的 Mandelbrot 图形。

假设选择绘图区域为[48] $x \in [-0.19050, -0.13824], y \in [1.01480, 1.06707]$，可以绘制出相应的 Mandelbrot 图，如图 1-2 所示。

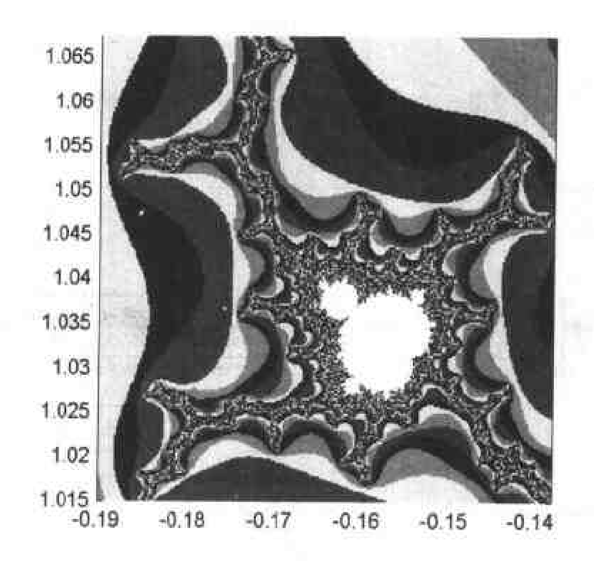


图 1-2 Mandelbrot 分形图

```
function examp2
x=linspace(-0.19050,-0.13824,300); % 选定横坐标范围
y=linspace(1.01480,1.06707,300);   % 选定纵坐标范围
[X,Y]=meshgrid(x,y); n_iter=200; c=0; a=2;
W=mandelbrot(X,Y,c,a,n_iter);  % 调用 Mandelbrot 映射程序
pcolor(X,Y,W), shading flat;
axis('square'); colormap prism(256);
```

```
function W=mandelbrot(X,Y,c,a,n_iter) % Mandelbrot 映射程序
C=X+i*Y; W=zeros(size(X)); Z=c;
for k=1:n_iter,
   Z=Z.^2+C;  % 复数映射
   i0=find(abs(Z)>a); W(i0)=k; Z(i0)=NaN;
end
i0=find(W==0); W(i0)=NaN;
```

〖例 1.6〗 **复杂系统框图绘制与仿真** —— 很多系统是以框图形式给出的，如图 1-3 中给出的直流电机拖动系统，用其他方式还是很不易仿真的，可以用 Simulink 搭建起如图 1-4 所示的框图，然后直接对之仿真，详细分析见例 4.2。

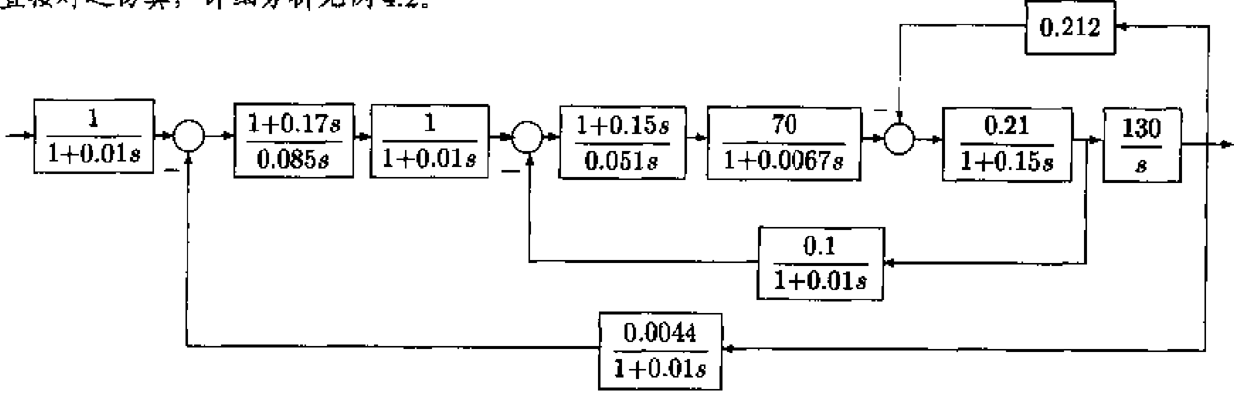


图 1-3 一个直流电机拖动系统的例子

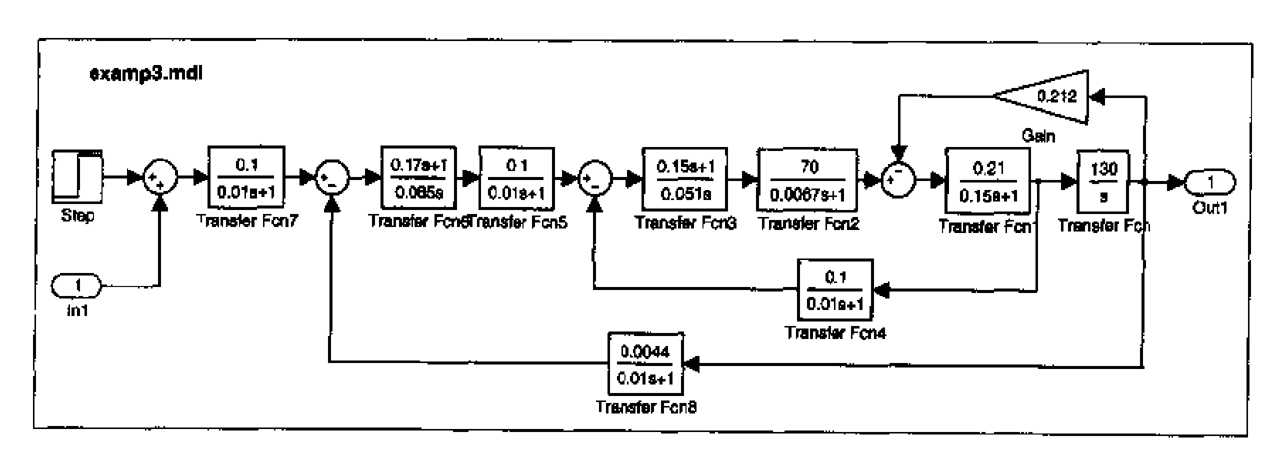


图 1-4 直流电机拖动模型 Simulink 模型表示

〖例 1.7〗 **异步电机系统仿真** —— 对电机内部信号的仿真实际上是更难的事情，好在 Simulink 有很实用的电力系统模块集，将很多的电机模型都构造成模块，用户可以直接对这些模块进行仿真分析，获得电机内部的信号。图 1-5 将给出三相交流电星型接法下异步电动机的仿真框图，用该框图可以直接对系统进行仿真，详细分析见例 5.23。启动仿真过程，就可以将转速、转矩和通常难以测得的转子、定子电流在“示波器 (Scope)”中显示出来。有关异步电动机的仿真的详细情况请参见例 5.23。

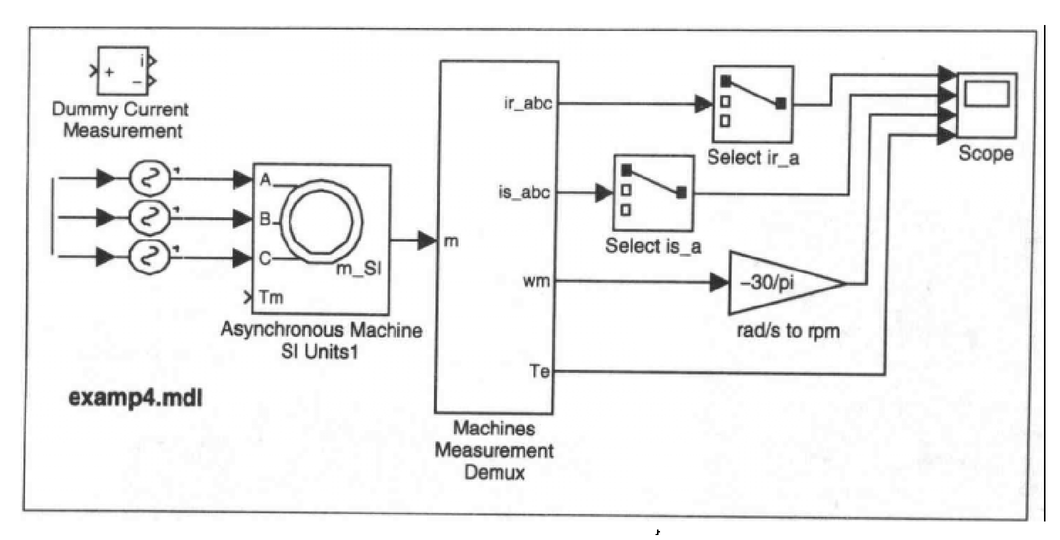


图 1-5 异步机仿真 Simulink 模型

在当前理工科专业的课程中，存在许多数值问题和解析问题的求解，而在现行教材中，往往由于没有实用计算机语言、软件和计算机辅助环节的支持，所以只能给出简单的、利于手工运算的问题的求解，而避开了可能要求较深数学运算的问题。另外在很多课程中需要引入实验环节，而很多高校目前的实验状态还不能达到令人满意的状态，所以可以考虑用 MATLAB/Simulink 语言环境建立起相关课程的“软实验”环境，提高教学水平。

〖例 1.8〗 **虚拟现实技术及其应用** —— MATLAB 的虚拟现实工具箱是随着 MATLAB 6.1 正式推出的，它使得虚拟现实技术的实现变得更容易。图 1-6 给出了一个飞机绕摩天大楼飞行的 Simulink 模型，其中调用了虚拟现实模块，运行该模型可以自动打开网页浏览器，观察该过程的虚拟现实效果。具体例子参见例 6.22 与 6.23。可以说，对虚拟现实技术的支持使得 MATLAB/Simulink 在系统仿真领域的应用上了一个新的台阶。

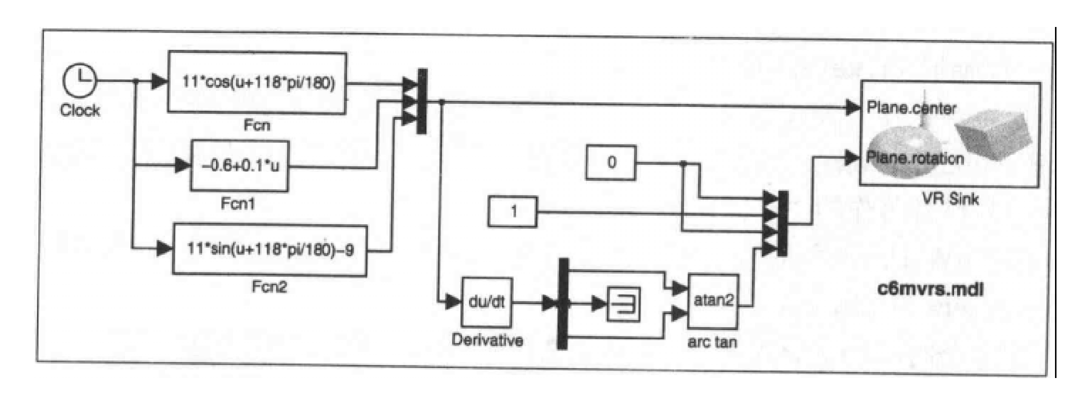


图 1-6 飞机飞行仿真模块框图

1.3.4 互联网上的 MATLAB 资源

“MATLAB 大观园”是作者致力维护的、国内最受欢迎的 MATLAB 网站，该网站上有大量的 MATLAB 资源，如很多免费工具箱的下载、MATLAB 讲座、作者发表文

章、著作中的资源文件下载等，其地址为

http://matlab.myrice.com

其界面如图 1-7 所示。另外，在该网站下还挂有“MATLAB 语言与应用”论坛，该论坛

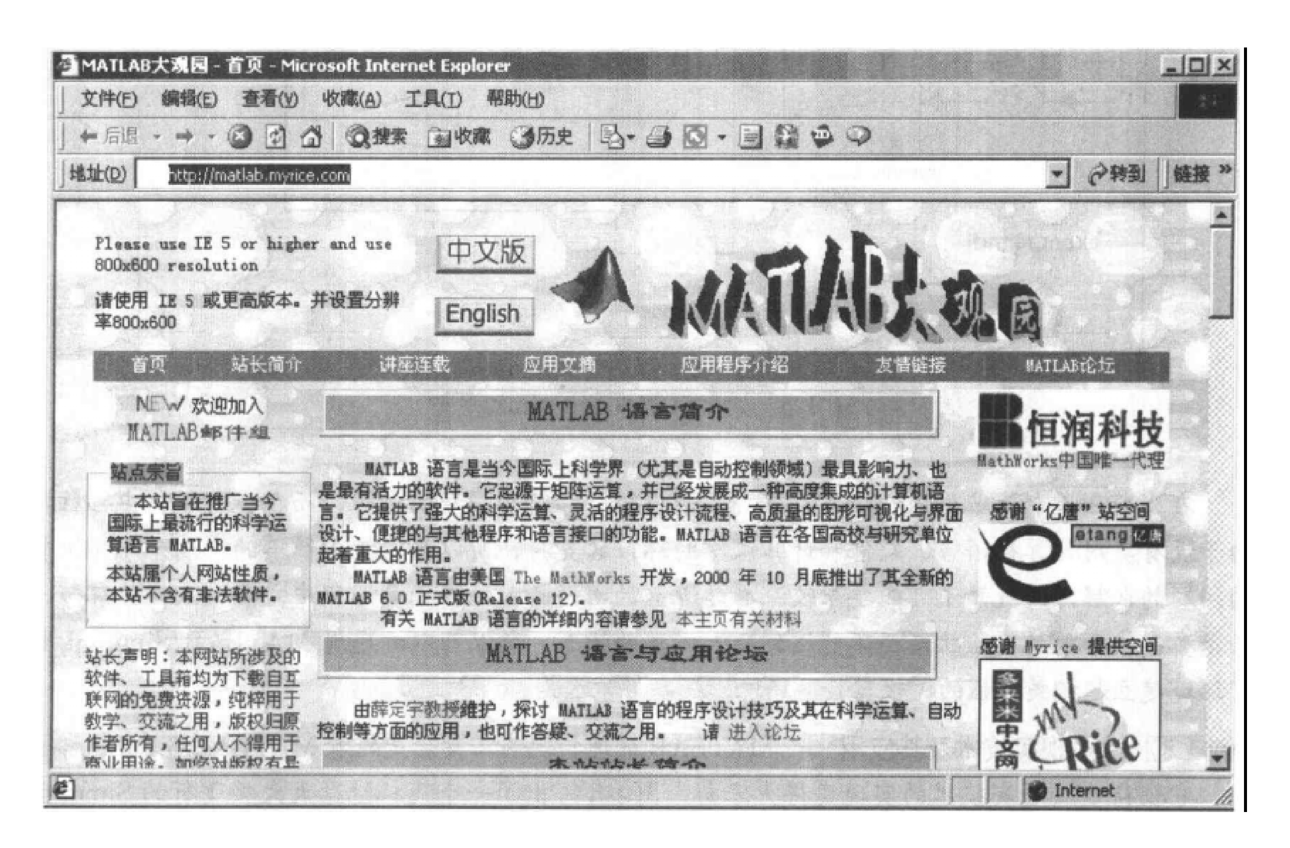


图 1-7 “MATLAB 大观园”界面

的地址为

http://matlab.netsh.net

该论坛是深受使用 MATLAB 语言的师生喜爱的论坛，在该论坛中经常有使用者交流 MATLAB/Simulink 编程的方法、问题，也经常有国内外的本领域专家解答问题，所以该网站被戏称为作者著作的“售后服务站”。

The MathWorks 公司的官方网站无疑有 MATLAB 发布的最新消息，其地址为

http://www.mathworks.com

该网站内容较全面，有全套 MATLAB 联机帮助文件和说明书的电子版免费下载，定期发布 News Letter 和文摘，The MathWorks 公司还办有 MATLAB 的讨论组，地址为

news://www.mathworks.com/comp.soft-sys.matlab

在该讨论组中经常有 MATLAB 的高手和相关领域的专家解答各种问题。

MATLAB Central 是 The MathWorks 公司新近推出的服务网址：

http://www.mathworks.com/matlabcentral/index.shtml

在该网站中，包含了原来 The MathWorks 原来用户支持网站的大部分内容，如用户开发的程序下载、用户组讨论区、帮助文件下载等，值得访问。其中讨论组的 Web 浏览

器网址现在为：

http://newsreader.mathworks.com

下面将给出其他有关的网址：

- **九州恒润公司网站** 作为国内 MATLAB 及相关产品的独家代理，恒润公司的网站及技术论坛也是很有特色的，其网址为：

  http://www.hirain.com

- **MathTools 网站** 前 MathTools 公司开发的 MATCOM 是将 MATLAB 语言程序翻译成 C/C++ 程序的强有力工具，该公司被 The MathWorks 吞并后，其原有的网站仍然是很有活力的，内容也是很充实的，值得访问：

  http://www.mathtools.net

- **"MATLAB Ring" 网站** 实际上相当于 MATLAB 网站搜索的一个好的引擎，该网站下分类列出了其他 MATLAB 网站，其网址为：

  http://www.sstreams.com/rings/matlab

- **哈工大 MATLAB 教学网站** 提供各种资源下载，如 MATLAB 电子手册、书籍的下载，国外优秀站点镜像，哈工大相关课程课件等。该网址地址为：

  http://matlab.turbo.hit.edu.cn

- **"MATLAB 乐园" 网站** 提供了很多资源的下载，该网站似乎更侧重于 MATLAB 语言与 C 接口方面的讨论，很有特色，并配有论坛。该网站的网址为：

  http://mathtools.top263.net

- 国内很多高校教育网的 BBS 都辟有 MATLAB 与应用方面的讨论区，较好的有清华大学、哈尔滨工业大学、西安交通大学、上海交通大学等网站等[①]。

1.4 本书的结构和代码

1.4.1 本书的结构

本书第 1 章 (本章) 中将首先介绍系统仿真的概念，并介绍数学软件包和仿真语言、科学运算语言的概述，并初步介绍并演示 MATLAB/Simulink 语言在系统仿真领域的应用；第 2 章将介绍 MATLAB 语言编程的基础知识，包括一般的数据结构、语句结构、函数编写、图形可视化和界面设计，并介绍 MATLAB 程序的加速技巧，旨在为读者提供关于 MATLAB 语言编程的必备知识；第 3 章中将系统介绍 MATLAB 语言求解数学问题的内容，包括数值线性代数运算、微积分的数值解法和解析解法、各类常微分方程的数值解法、 最优化问题的解法和数据分析的有关内容；第 4 章将初步介绍各种 Simulink 模块组，并介绍 Simulink 环境的基本使用方法，然后举例演示 Simulink 在数学建模中的应用。该章还将介绍线性系统的仿真方法、随机激励下连续系统的仿真方法和结果分析、

[①] 由于本人访问过的网站有限，难免有疏漏，望没有提及的网站谅解。另外，上述内容不具排名意义。

分形系统的仿真与图形显示等内容；第5章中将介绍 Simulink 仿真的进一步知识，如常用的模块应用技巧、各种各样的模块输出方法、子系统概念和模块的封装技术，然后将介绍两个实用的 Simulink 工具集，如非线性控制设计工具集及其在控制参数优化设计中的应用，电力系统模块集在一般电路仿真、电子电力系统仿真和异步机仿真中的应用，最后还将给出一个内燃机系统仿真的个案分析例子；第6章将介绍 Simulink 环境的高级应用技术，如利用 MATLAB 语句的 Simulink 建模与仿真方法、非线性模型的线性化技术、S-函数的 MATLAB 程序格式和 C 程序格式及应用、Stateflow 建模与应用及虚拟现实工具箱的应用技术；第7章中将介绍半实物仿真与实时控制技术。在附录中给出书中作者编写的 MATLAB 函数和 Simulink 框图，这些代码量较大，不适于用户重新键入，为方便读者，将其放置在很多网站下直接下载，例如在 MATLAB 大观园中就可以直接下载本书和作者其他书籍中的程序。

作者编写的函数、Simulink 模型和其他文件在附录中列出，其内容全部存储在 simbook.zip 文件中，所有例题中执行的语句存储在 simexamp.zip，均按例题编号存储，下载网址分别为：

```
http://matlab.myrice.com/simbook.zip
http://matlab.myrice.com/simexamp.zip
```

用户可以分别载、解压这两个文件，并将它们所在的目录设置到 MATLAB 的运行目录下，这样就可以直接运行本书中的文件了。

1.4.2 书中英文字体说明

在这里我们对书中所用的字体作整体的说明，相信这些字体的使用对读者正确理解本书的内容有所帮助。

- 英文文体与公式中常量采用正体 Times-Roman 字体，如 MATLAB, e, x 轴。
- 公式中变量字体采用 Times-Roman 斜体字，如 x, t 等；矩阵、向量名用黑斜体，如 $\boldsymbol{A}$, $\boldsymbol{x}$，$\boldsymbol{f}(t,\boldsymbol{x})$ 等。
- 程序清单、函数、命令语句及函数中变量名等采用 Courier 正体，如 `eig()`，`tspan`，`simulink3` 等。
- 界面中的文字标识、Simulink 模块的名称等采用 Sans Serif 正体，如 File 菜单，OK 按钮、Step 模块等。

1.5 习 题

(1) MATLAB 提供了较好的演示程序，在 MATLAB 的命令窗口下键入 `demo` 则可以直接运行演示程序，试运行 MATLAB 的演示程序，初步领略 MATLAB 的强大功能。

(2) 本书中给出了大量作者编写的程序，在深入学习本书的内容前应该从互联网上下载这些文件．本书介绍的全部程序都是可重复的，实践这些程序将有助于更好地理解本书介绍的内容，这些文件都可以从作者维护的"MATLAB 大观园"下载 (`http://matlab.myrice.com`)。

第 2 章 MATLAB 语言程序设计基础

前面综述了用于计算机仿真的各种计算机语言，当前国际上科学运算与计算机仿真领域首选的计算机语言为 MATLAB 语言，我们将在本章中较全面地介绍 MATLAB 语言的编程方法与技巧，首先在第 2.1 节中介绍 MATLAB 语言环境操作界面与使用的有关内容，在第 2.2 节中将介绍 MATLAB 下支持的数据结构，首先介绍传统的矩阵结构，然后将介绍更复杂的结构。第 2.3 节将介绍矩阵的代数运算、逻辑运算、比较运算和其他基本运算。第 2.4 节中将介绍各种语句流程结构，如循环结构、转移结构、开关结构和 MATLAB 特有的试探结构。第 2.5 节将较深入地介绍 MATLAB 编程的关键技术，即 MATLAB 函数的编写及技巧，并介绍 M 函数的跟踪调试技术。第 2.6 和 2.7 节将介绍 MATLAB 的绘图方法，包括二维图、三维图和图像绘制技术，并将介绍图形的各种修饰方法。第 2.8 节中介绍的 MATLAB 图形用户界面设计技术将使得用户掌握新的方法，方便地给自己的程序设计出优美、友好的图形界面来。由于 MATLAB 程序的执行效率一般低于低级编程语言，所以在第 2.9 节中将介绍 MATLAB 程序的加速技巧，包括“向量”法编程思路及 Mex 函数的编写方法。

2.1 MATLAB 语言的基本使用环境

2.1.1 MATLAB 语言界面

MATLAB 6.1 版本包含两张光盘，将其中的程序盘插入驱动器，运行其上的安装文件 (setup.exe)，则可以按提示安装整个 MATLAB 系统。MATLAB 6.0 版本和中文版 Windows ME, NT 和 2000 在字库上有点小冲突，所以应该下载一个补丁文件，覆盖 MATLAB 路径下的 `system\java\jre\win32\jre\lib` 中的相应文件即可，该文件可以从 “MATLAB 大观园” 直接下载 (`http://matlab.myrice.com/progs/matlabfonts.zip`)。

MATLAB 6.1 解决了字体问题，所以在中文版 Windows 环境下再安装 MATLAB 不会出现上述的问题，故没有必要再下载上述的补丁了。

和以前的版本相比，MATLAB 6.x 在界面上的变化是很大的，以前的版本只给出一个命令窗口，MATLAB 6.1 的程序界面如图 2-1 所示，除了其右侧的命令子窗口之外，还有在前台的 Launch Pad (程序调用板) 和 Command History (命令的历史记录) 两个子窗口，以及后台的 Workspace (工作空间管理程序) 和 Current Directory (当前目录管理程序) 等，这从一定程度上使得 MATLAB 本身的操作更容易、方便了。

2.1.2 MATLAB 的联机帮助与电子版手册

在 MATLAB 的文档光盘 (Documentation CD) 中，包含了几乎全套工具箱使用

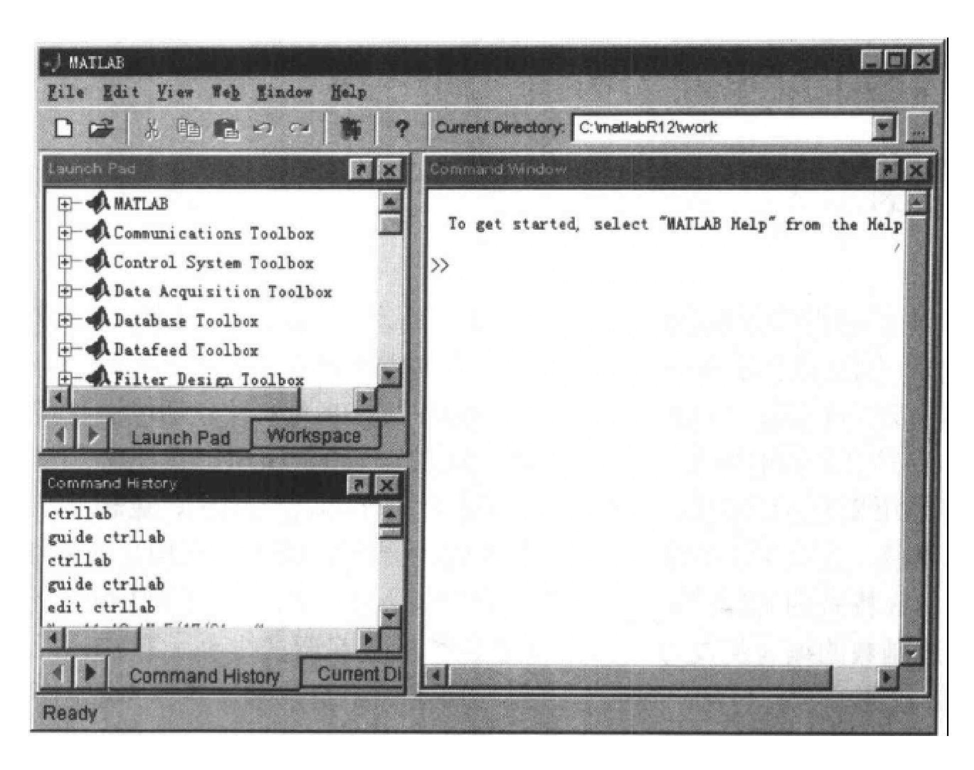


图 2-1　MATLAB 6.1 程序界面

手册的 HTML 和 PDF 版本文件，其中 PDF 文件可以由一个免费的 Acrobat Reader 程序阅读，该阅读软件可以从很多软件下载网站，包括作者维护的“MATLAB 大观园” (`http://matlab.myrice.com`) 中免费下载。在 Acrobat Reader 下打开电子版的 MATLAB 手册，则得出如图 2-2 所示的界面，利用该阅读软件提供的强大功能，则可以方便地阅读有关手册，有时阅读这样的手册比一些印刷的手册更方便，因为它支持超文本链接等功能。

将 MATLAB 文档光盘插入光驱，则在 MATLAB 命令窗口中选择 Help | MATLAB Help 菜单项，则将得出如图 2-3 所示的联机帮助窗口界面，用户可以从该窗口出发得出有关的帮助信息帮助，也可以选中的功能时按下 F1 键启动即可。

2.2　MATLAB 语言的数据结构

强大方便的数值运算功能是 MATLAB 语言的最显著特色，从计算精度出发，MATLAB 下最常用的数值量为双精度浮点数，占 8 个字节 (64 位)，遵从 IEEE 记数法，有 11 个指数位、53 位尾数及一个符号位，值域的近似范围为 -1.7×10^{308} 至 1.7×10^{308}，其 MATLAB 表示为 `double()`。考虑到一些特殊的应用，比如图像处理，MATLAB 语言还引入了无符号的 8 位整形数据类型，其 MATLAB 表示为 `uint8()`，其值域为 0 至 255，这样可以大大地节省 MATLAB 的存储空间、提高处理速度。此外，在 MATLAB 中还可以使用其他的数据类型：`int8()`, `int16()`, `int32()`, `uint16()`, `uint32()` 等，每一个类型后面的数字表示其位数，其涵义不难理解。

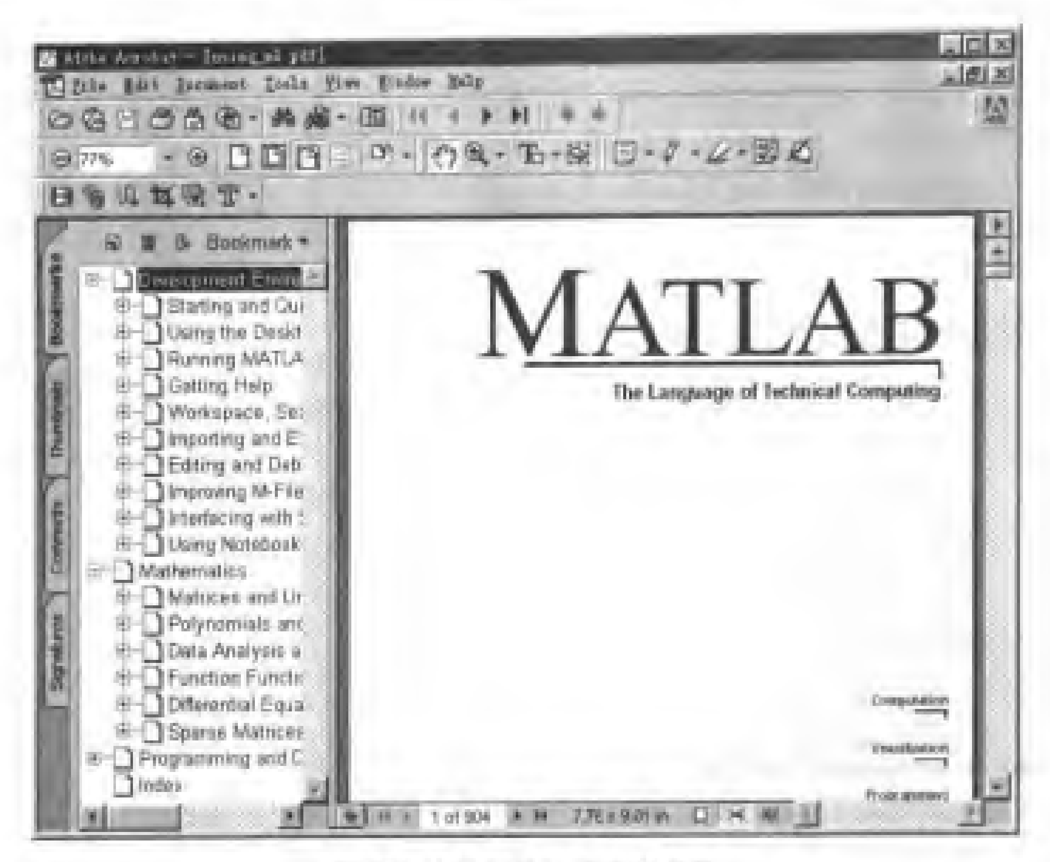


图 2-2 MATLAB 电子版手册

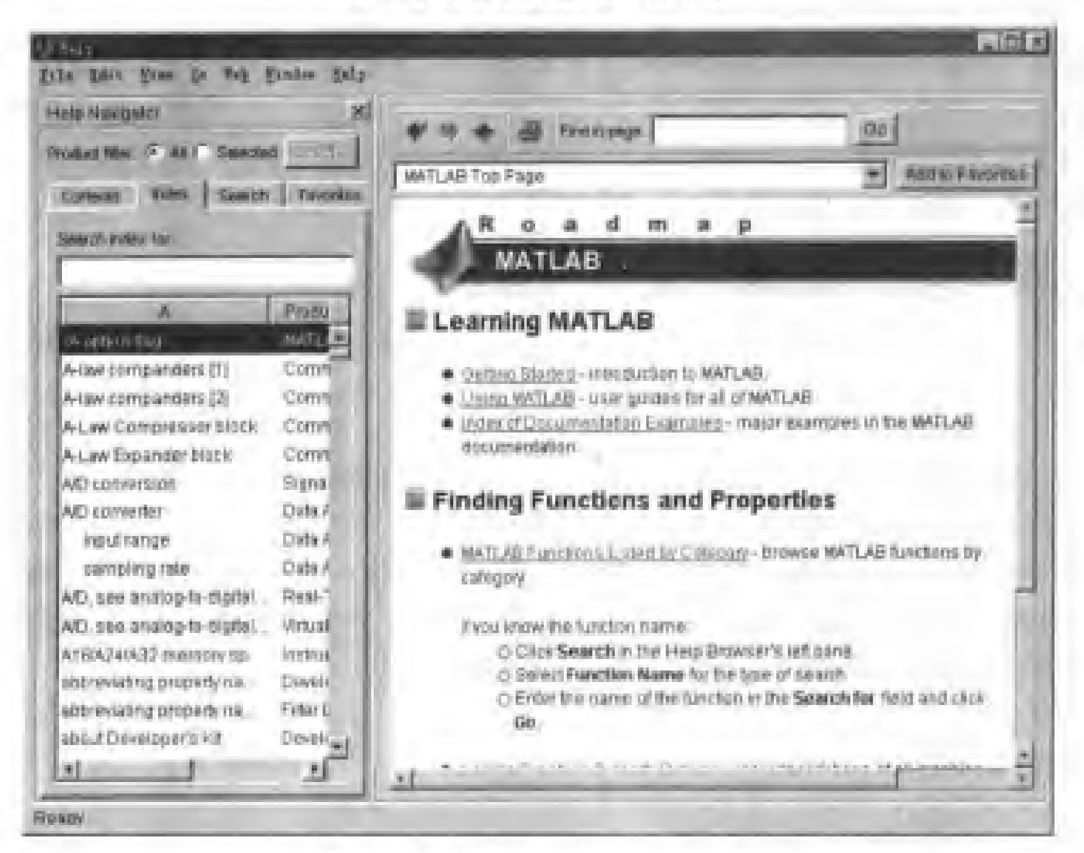


图 2-3 MATLAB 的联机帮助界面

除了一般的实数数据之外，MATLAB 还支持复数向量、矩阵和字符串型矩阵，从 MATLAB 5.0 版本开始，还允许其他更高级的数据类型，如多维数组、数据结构体、单元数据、类和对象等，这使得 MATLAB 编程功能变得更强大。本节首先介绍 MATLAB 下的常量、变量与赋值语句，然后将介绍最常用的矩阵表示，最后将再介绍其他的数据结构内容及其应用。

2.2.1 常量、变量与赋值语句

MATLAB 语言变量名应该由一个字母引导，后面可以跟字母、数字、下划线等。例如，MYvar12, MY_Var12 和 MyVar12_ 均为有效的变量名，而 12MyVar 和 _MyVar12 为无效的变量名。在 MATLAB 中变量名是区分大小写的，就是说，Abc 和 ABc 两个变量名表达的是不同的变量。所以在使用 MATLAB 语言编程时一定要注意。

MATLAB 的赋值语句有下面两种结构：

(1) **直接赋值语句**

直接赋值语句的基本结构如下：

```
赋值变量= 赋值表达式
```

这一过程把等号右边的表达式直接赋给左边的赋值变量，并返回到 MATLAB 的工作空间。如果赋值表达式后面没有分号，则将在 MATLAB 命令窗口中显示表达式的运算结果。若不想显示运算结果，则应该在赋值语句的末尾加一个分号。如果省略了赋值变量和等号，则表达式运算的结果将赋给保留变量 ans。所以说，保留变量 ans 将永远存放最近一次无赋值变量语句的运算结果。

(2) **函数调用语句**

函数调用语句的基本结构为：

```
[返回变量列表] = 函数名(输入变量列表)
```

其中函数名的要求和变量名的要求是一致的，一般函数名应该对应在 MATLAB 路径下的一个文件，例如，函数名 my_fun 应该对应于 my_fun.m 文件。从 MATLAB 5.0 版本开始，允许该函数名为当前函数文件中的一个子函数。当然，还有一些函数名需对应于 MATLAB 内核中的内在 (built-in) 函数，如 inv() 函数等。

返回变量列表和输入变量列表均可以由若干个变量名组成，它们之间应该分别用逗号，返回变量还允许用空格分隔，例如 [U,S,V] = svd(X)，该函数对给定的 X 矩阵进行奇异值分解，所得的结果由 U, S, V 三个变量返回。如果不想显示函数调用的最终结果，在函数调用语句后仍应该加个分号，如 [U S V] = svd(X);。

在 MATLAB 语言中还为特定常数保留了一些名称，虽然这些常量都可以重新赋值，但建议您在编程时应尽量避免对这些量重新赋值。

- eps — 机器的浮点运算误差限。PC 机上 eps 的默认值为 2.2204×10^{-16}，若某个量的绝对值小于 eps，则可以认为这个量为 0。

- i 和 j　若 i 或 j 量不被改写，则它们表示纯虚数量 i。但在 MATLAB 程序编写过程中经常事先改写这两个变量的值，如在循环过程中常用这两个变量来表示循环变量，所以应该确认使用这两个变量时没有被改写。如果想恢复该变量，则可以用下面的形式设置：i=sqrt(-1)，即对 -1 求平方根。
- Inf —无穷大量 $+\infty$ 的 MATLAB 表示，也可以写成 inf。同样地，$-\infty$ 可以表示为 -Inf。在 MATLAB 程序执行时，即使遇到了以 0 为除数的运算，也不会终止程序的运行，而只给出一个"除 0"警告，并将结果赋成 Inf，这样的定义方式符合 IEEE 的标准。从数值运算编程角度看，这样的实现形式明显优于 C 这样的非专用语言。
- NaN —不定式 (Not a Number)，通常由 0/0 运算、Inf/Inf 及其他可能的运算得出。NaN 是一个很奇特的量，如 NaN 与 Inf 的乘积仍为 NaN。
- pi — 圆周率 π 的双精度浮点表示。
- lasterr — 存放最新一次的错误信息。此变量为字符串型，如果在本次执行过程中没出现过错误，则此变量为空字符串。
- lastwarn — 存放最新的警告信息。若未出现过警告，则此变量为空字符串。

2.2.2 矩阵的 MATLAB 表示

从 MATLAB 的名字就可以看出，它最基本的出发点是一个"矩阵实验室"，即它是以矩阵为基本变量单元的。确切地说，它是以复数矩阵为最基本的变量单元的。当然从 MATLAB 5.0 开始，还引入了更复杂的数据结构，使得 MATLAB 编程更加灵活。这里先介绍 MATLAB 传统的变量类型。

在 MATLAB 语言中表示一个矩阵是很容易的事，例如，矩阵

$$\boldsymbol{A}=\begin{bmatrix}1&2&3\\4&5&6\\7&8&0\end{bmatrix}$$

可以由下面的 MATLAB 语句直接输入到工作空间中。

```
>> A=[1,2,3; 4 5,6; 7,8 0]
A =
        1     2     3
        4     5     6
        7     8     0
```

其中的 >> 为 MATLAB 的提示符，由机器自动给出，在提示符下可以输入各种各样的 MATLAB 命令。

矩阵的内容由方括号括起来的部分表示，而在方括号中的分号表示矩阵的换行，逗号或空格表示同一行矩阵元素间的分隔。给出了上面的命令，就可以在 MATLAB 的工作空间中建立一个 A 变量了。

在 MATLAB 编程中有一个约定，如果在一个赋值语句后面没有分号，则等号左边的变量将在 MATLAB 命令窗口中显示出来，显示的格式如前面所示。如果不想显示中间结果，则应该在语句末尾加一个分号，如：

```
>> A=[1,2,3; 4 5,6; 7,8 0]; % 不显示结果，但进行赋值
```

在 MATLAB 下也可以容易地输入向量和标量。例如，行向量和列向量可以分别由下面两条命令直接输入：

```
>> V1=[1 2 3,4]  % 行向量输入
V1 =
     1    2    3    4
>> V2=[1; 2; 3; 4]   % 列向量输入
V2=
     1
     2
     3
     4
```

除了表示向量和矩阵之外，利用 MATLAB 还可以容易地表示标量。学会了矩阵的基本表示方法之后，就可以容易地理解下面赋值表达式的方式和结果了。

```
>> A=[A; [1 3 5]]  % 在矩阵后面再补一行
A =
    1    2    3
    4    5    6
    7    8    0
    1    3    5
```

MATLAB 语言定义了独特的冒号表达式来给行向量赋值，其基本使用格式为：

```
a=s1:s2:s3;
```

其中，s1 为起始值，s2 为步距，s3 为终止值。如果 s2 的值为负值，则要求 s1 的值大于 s3 的值，否则结果为一个空向量 a。如果省略了 s2 的值，则步距取默认值 1。例如，前面的 V1 行向量可以由 V1=1:4 语句赋值。

可以通过下面的语句定义一个行向量：

```
>> a=0:0.1:1.16
a =
  Columns 1 through 7
        0    0.1000    0.2000    0.3000    0.4000    0.5000    0.6000
  Columns 8 through 12
   0.7000    0.8000    0.9000    1.0000    1.1000
```

复数矩阵的输入同样也是很简单的，在 MATLAB 环境中定义了两个记号 i 和 j，可

以用来直接输入复数矩阵。例如，如果想在 MATLAB 环境中输入矩阵

$$B=\begin{bmatrix}1+9\mathrm{i} & 2+8\mathrm{i} & 3+7\mathrm{i}\\4+6\mathrm{i} & 5+5\mathrm{i} & 6+4\mathrm{i}\\7+3\mathrm{i} & 8+2\mathrm{i} & 0+\mathrm{i}\end{bmatrix}$$

则可以通过下面的 MATLAB 语句直接进行赋值。

```
>> B=[1+9i,2+8i,3+7j; 4+6j 5+5i,6+4i; 7+3i,8+2j 1i]
B =
   1.0000 + 9.0000i   2.0000 + 8.0000i   3.0000 + 7.0000i
   4.0000 + 6.0000i   5.0000 + 5.0000i   6.0000 + 4.0000i
   7.0000 + 3.0000i   8.0000 + 2.0000i        0 + 1.0000i
```

2.2.3 多维数组的定义

除了标准的二维矩阵之外，MATLAB 从 5.0 版本开始定义三维或多维数组。三维数组很好理解，假设有若干个维数相同的矩阵 $\boldsymbol{A}_1,\boldsymbol{A}_2,\cdots,\boldsymbol{A}_m$，那么把这若干个矩阵一页一页地叠起来，就可以构成一个三维数组。三维数组的示意图如图 2-4 所示。假设可以如

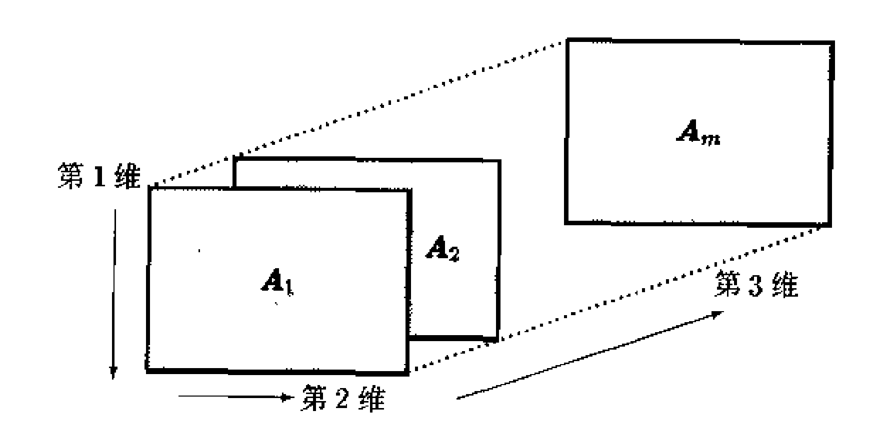


图 2-4 三维数组的示意图

下定义 A1, A2, A3 矩阵：

```
>> A1=[1,2,3; 4 5 6; 7 8,9]; A2=A1'; A3=A1-A2;
```

则通过下面最原始的方法就可以定义出一个三维数组 A4

```
>> A4(:,:,1)=A1; A4(:,:,2)=A2; A4(:,:,3)=A3
A4(:,:,1) =
     1     2     3
     4     5     6
     7     8     9
A4(:,:,2) =
     1     4     7
```

```
     2     5     8
     3     6     9
A4(:,:,3) =
     0    -2    -4
     2     0    -2
     4     2     0
```

MATLAB 语言提供了一个 cat() 函数来构造多维数组。该函数的调用格式为：

```
A=cat(n,A1,A2,···, Am)
```

其中，n=1 和 2 时分别构造 [A1;A2;...; Am] 和 [A1,A2,...,Am]，结果是二维数组，而 n=3 可以构造出三维数组。例如前面原始的命令可以由下面的简单函数调用语句取代。

```
>> A5=cat(3,A1,A2,A3)
A5(:,:,1) =
     1     2     3
     4     5     6
     7     8     9
A5(:,:,2) =
     1     4     7
     2     5     8
     3     6     9
A5(:,:,3) =
     0    -2    -4
     2     0    -2
     4     2     0
```

比较下面两个语句获得的二维矩阵。

```
>> A6=cat(2,A1,A2,A3)
A6 =
     1     2     3     1     4     7     0    -2    -4
     4     5     6     2     5     8     2     0    -2
     7     8     9     3     6     9     4     2     0
>> A7=cat(1,A1,A2,A3)
A7 =
     1     2     3
     4     5     6
     7     8     9
     1     4     7
     2     5     8
```

```
     3     6     9
     0    -2    -4
     2     0    -2
     4     2     0
```

一个多维数组 (包括矩阵) 的大小可以由 size() 函数来测出。

```
>> size(A)    % 显示矩阵的行数和列数
ans =
     4     3
>> size(A4)  % 显示多维数组的各维大小
ans =
     3     3     3
```

2.2.4 数据结构体

数据结构体将某一类相关的信息纳入一个统一的变量名字下进行管理。比如说要建立学生档案就可以采用数据结构体。事实上，结构体可以理解成在 MATLAB 语言下的一个小型的"数据库"。下面将通过一个例子来说明结构体的建立、修改及应用。

〖例 2.1〗 *考虑建立学生档案结构体，这里面应包含下列信息：*

- *编号 (用 number 表示，而在结构体下 number 又称为其成员变量或域)：数值型。*
- *姓名 (name)：字符串型。*
- *身高 (height)：数值型。*
- *考试成绩 (test)：矩阵，其第 i 行为第 i 次考试的成绩，而第 j 列为第 j 门考试的成绩。*

有了上述要求，就可以由下面的语句建立起结构体 student_rec:

```
>> student_rec.number=1;
   student_rec.name='张三';
   student_rec.height=180;
   student_rec.test=[100, 80, 75; 77, 60, 92;
                     67, 28, 90; 100, 89, 78];
   student_rec      % 显示结构体的内容
student_rec =
    number: 1
      name: '张三'
    height: 180
      test: [4x3 double]
```

和 C 语言不同，引用结构体中成员变量时直接引用即可，而无需用 -> 算符来引用。例如若想获得该结构体中的 test 成员变量，则可以通过下面的语句来直接得出。

```
>> student_rec.test
ans =
```

```
   100    80    75
    77    60    92
    67    28    90
   100    89    78
```

下面的命令建立一组 100 个学生 (假设 2 个年级，每个年级 50 人) 的新的结构体变量 b:

```
>> b(50,2)=struct(student_rec)  % 构造 50x2 结构体矩阵
b =
   50x2 struct array with fields:
     number
     name
     height
     test
```

这样就可以构成一个小型的数据库了。数据库中每个条目的具体内容还需要根据实际情况填写，填写的格式是很简单的，例如第 2 年级第 43 个学生的有关信息可以由下面的语句填写。

```
>> b(43,2).number=50+43;
   b(43,2).name='李四';
   b(43,2).height=186;
   b(43,2).test=[83, 80, 78; 97, 80, 72;
                 69, 88, 80; 87, 99, 100];
```

结构体的结构也是可以修改的，例如若想给 b 变量加一个成员变量 —— 体重 (weight)，则可以将结构体中任意一个变量后面加一个 weight 成员变量，即

```
>> b(1,1).weight=90   % 添加成员变量 weight
b =
   50x2 struct array with fields:
     number
     name
     height
     test
     weight
```

这样就可以把所有的 b 结构体变量均加一个 weight 成员变量，但每个其他成员函数均为空矩阵，可以使用赋值语句给它赋予确定的值。

如果想消除结构体变量中某一成员变量，则可以使用 rmfield() 函数来完成。例如，如果想消除 b 结构体中的 weight 成员变量，则可以给出如下语句：

```
>> b=rmfield(b,'weight')  % 输出成员变量 weight
b =
   50x2 struct array with fields:
     number
     name
```

```
height
test
```

2.2.5 单元结构

MATLAB 从 5.0 版开始引入了一种崭新的数据结构，称为单元 (cell) 结构。该结构把不同属性的数据都纳入到一个变量之下，而这个变量称为单元。单元结构的概念有些类似于结构体。所不同的是，结构体下的各个子项称为成员变量，而每个成员变量都有自己的名字。单元变量的表示方法更类似于带有下标的矩阵和多维数组，但这些下标不是用圆括号括起来，而是用大括号括起来。在矩阵和多维数组中，每个矩阵元素都应该具有相同的属性，而单元结构则没有此要求，用户可以把各种不同属性的数据全部归并到一个单元变量中。

〖例 2.2〗 这里重新考虑例 2.1 中提出的问题，可以用单元数据结构来构造某个学生的档案。

```
>> B={1,'张三',180,...
    [100, 80, 75; 77, 60, 92; 67, 28, 90; 100, 89, 78]}
B =
   [1]    '张三'     [180]    [4x3 double]
>> size(B)
ans =
     1     4
```

其中 ... 为续行符号，表示下一行的内容紧跟上行，这样就可以将长的 MATLAB 语句分成几行来表示。可见，对单元变量也可以使用 size() 和 length() 函数。从上面的显示中，并不能直接看出此单元数据的第 4 个元素代表的矩阵具体内容是什么，只能够定性地看出它是一个 4×3 矩阵。

可以由带有大括号下标的形式访问该单元数据，得出该矩阵的具体内容为：

```
>> B{4}     % 显示第 4 单元内容
ans =
   100    80    75
    77    60    92
    67    28    90
   100    89    78
```

注意，不能用圆括号去访问单元数组。另外，上面得出的 ans 结果不再是单元数组，而是常规的矩阵了。也可以使用 celldisp() 函数来直接显示整个单元。

```
>> celldisp(B)  % 显示整个单元变量
B{1} =
     1
B{2} =
   张三
```

```
B{3} =
     180
B{4} =
    100    80    75
     77    60    92
     67    28    90
    100    89    78
```

可以消除单元变量中的某个单元元素。例如，如果想删除 B 单元变量的第 3 个单元元素，则可以给出下面的命令：

```
>> B(3)=[]
B =
    [1]    '张三'    [4x3 double]
```

可见，通过这个命令就消除了原单元数据 B 的第 3 个单元。注意，这里用的是 B(3) 而不是 B{3}，后者只能将第 3 个单元置成空矩阵，而不能消除它，所以这里的表示方法是 MATLAB 的约定，不一定有其他合适的解释。

单元数据、结构体与对象数据是可以相互嵌套的，这就给 MATLAB 的程序设计带来了极大的方便。灵活地使用各种数据结构，将有助于掌握 MATLAB 语言，解决自己的问题。

2.2.6 MATLAB 下的类与对象

我们知道，类 (class) 与对象 (object) 的概念是面向对象程序设计的基础。从前面结构体的叙述中可以看出，其成员都是以变量的形式出现的。类的概念是结构体的拓展，在类中，不但可以包含变量型的成员，还可以包含与这些变量相关联的函数或运算。对象是指类的一个实例。

在编程中使用类和对象的概念，将使您的程序变得更简单，并且可以使您的程序代码增加再应用的机会。比如可以把您的代码直接嵌入到一个新的程序中，而无需对它做太多的修改。限于篇幅，本书仅介绍类与对象的使用，而不涉及太多的内部构造与编程方法，有关内容请参见文献 [24, 54]。

2.3 MATLAB 下矩阵的运算

2.3.1 矩阵的代数运算

如果一个矩阵 $\boldsymbol{A}$ 有 n 行、m 列元素，则称 $\boldsymbol{A}$ 矩阵为 $n \times m$ 矩阵；若 $n = m$，则矩阵 $\boldsymbol{A}$ 又称为方阵。MATLAB 语言中定义了下面各种矩阵的基本代数运算。

- **矩阵转置**

在数学公式中一般把一个矩阵的转置记作 $\boldsymbol{A}^T$，假设 $\boldsymbol{A}$ 矩阵为一个 $n \times m$ 矩阵，则其转置矩阵 $\boldsymbol{B}$ 的元素定义为 $b_{ji} = a_{ij}$, $i = 1, \cdots, n$, $j = 1, \cdots, m$，故 $\boldsymbol{B}$ 为 $m \times n$ 矩阵。

例如，从下面给出的 $\boldsymbol{A}$ 矩阵可以容易地求出其转置矩阵 $\boldsymbol{A}^T$。

$$\boldsymbol{A}=\begin{bmatrix}1&2&3\\4&5&6\\7&8&0\end{bmatrix},\quad \boldsymbol{A}^T=\begin{bmatrix}1&4&7\\2&5&8\\3&6&0\end{bmatrix}$$

如果 $\boldsymbol{A}$ 矩阵含有复数元素，则对之进行转置时，其转置矩阵 $\boldsymbol{B}$ 的元素定义为 $b_{ji}=a_{ij}^*, i=1,\cdots,n, j=1,\cdots,m$，亦即首先对各个元素进行转置，然后再逐项求取其共轭复数值。这种转置方式又称为 Hermit 转置，其数学记号为 $\boldsymbol{b}=\boldsymbol{A}^*$。例如下面给定复数矩阵及其转置分别为

$$\boldsymbol{A}=\begin{bmatrix}5+\mathrm{i}&2-\mathrm{i}&1\\6\mathrm{i}&4&9-\mathrm{i}\end{bmatrix},\ \boldsymbol{A}^*=\begin{bmatrix}5-\mathrm{i}&-6\mathrm{i}\\2+\mathrm{i}&4\\1&9+\mathrm{i}\end{bmatrix},\ \boldsymbol{A}^T=\begin{bmatrix}5+\mathrm{i}&6\mathrm{i}\\2-\mathrm{i}&4\\1&9-\mathrm{i}\end{bmatrix}$$

在 MATLAB 下，矩阵 $\boldsymbol{A}$ 的 Hermit 转置可以简单地由 A' 求出，

```
>> A=[5+i, 2-i, 1; 6*i, 4, 9-i]; B=A'   % 共轭转置
ans =
      5.0000 - 1.0000i          0 - 6.0000i
      2.0000 + 1.0000i     4.0000
      1.0000               9.0000 + 1.0000i
```

实数矩阵的转置也可以由简单命令 A' 求出。复数矩阵的常规转置 $\boldsymbol{A}^T$ 则可以调用 transponse() 函数来获得。

```
>> transpose(A)    % 直接转置
ans =
      5.0000 + 1.0000i          0 + 6.0000i
      2.0000 - 1.0000i     4.0000
      1.0000               9.0000 - 1.0000i
```

复数矩阵的 Hermit 转置还可以由 ctranspose(A) 实现，所以从等效的意义上讲，A'=ctranspose(A)，且 ctranspose(A)=transpose(conj(A))，其中 conj() 是求取共轭复数的函数。对实数矩阵来说二者没有区别。值得指出的是，conj() 函数还可以用于多维数组。

- **加减法运算**

假设在 MATLAB 工作环境下有两个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，则可以由下面的命令执行矩阵加减法：C=A+B 和 C=A-B。若 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的维数相同，它会自动地将 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的相应元素相加减，从而得出正确的结果，并赋给 $\boldsymbol{C}$ 变量。若二者之一为标量，则可以将其遍加 (减) 于另一个矩阵，在其他情况下，则 MATLAB 将自动地给出错误信息，提示用户两个矩阵的维数不匹配。

除了直接地使用 + 和 - 号进行运算之外，两个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的加减法运算还可以通过函数 plus(A,B) 和 minus(A,B) 来实现。

很多新的数据结构都定义了它自己的加减法运算。例如，这样的运算可以方便地用于多维数组等数据结构中。

• **矩阵乘法**

假设有两个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，其中 $\boldsymbol{A}$ 的列数与 $\boldsymbol{B}$ 矩阵的行数相等，或其一为标量，则称 $\boldsymbol{A}$, $\boldsymbol{B}$ 矩阵是可乘的，或称 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的维数是相容的。假设 $\boldsymbol{A}$ 为 $n \times m$ 矩阵，而 $\boldsymbol{B}$ 为 $m \times r$ 矩阵，则 $\boldsymbol{C} = \boldsymbol{AB}$ 为 $n \times r$ 矩阵，其各个元素为

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}, \text{ 其中 } i = 1, 2, \cdots, n, \ j = 1, 2, \cdots, r \tag{2.1}$$

例如下面给出的 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵及其乘积矩阵 $\boldsymbol{C}$ 为

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \ \boldsymbol{B} = \begin{bmatrix} 5 & 5 \\ 7 & 8 \end{bmatrix}, \ \boldsymbol{C} = \begin{bmatrix} 19 & 21 \\ 43 & 47 \end{bmatrix}$$

在 MATLAB 下，矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的乘积可以简单地由运算式 `C=A*B` 求出

```
>> A=[1, 2; 3, 4]; B=[5, 5; 7, 8]; C=A*B
C =

    19    21
    43    47
```

在这里并不需要指定 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的维数。如果 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的维数相容，则可以准确无误地获得乘积矩阵 $\boldsymbol{C}$；如果二者的维数不相容，则将给出错误信息，通知用户两个矩阵是不可乘的。

矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的乘法还可以由 `mtimes(A,B)` 来实现。

• **矩阵的左除**

MATLAB 定义了除法运算，它涉及到矩阵的求逆运算。MATLAB 还定义了矩阵的左除及右除。在这里只给出一般的等效关系，$\boldsymbol{A}$ 矩阵的逆还可以由 `inv(A)` 语句直接求出。

MATLAB 中用 "\" 运算符号表示两个矩阵的左除，`A\B` 为方程 $\boldsymbol{AX} = \boldsymbol{B}$ 的解 $\boldsymbol{X}$，若 $\boldsymbol{A}$ 为非奇异方阵，则 $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}$。如果 $\boldsymbol{A}$ 矩阵不是方阵，也可以求出 `A\B`，这时将使用最小二乘解法来求取 $\boldsymbol{AX} = \boldsymbol{B}$ 中的 $\boldsymbol{X}$ 矩阵。用这种方法得出的解可能与使用 `pinv()` 函数得出伪逆后再求出的结果不一致，所以使用时应该注意。MATLAB 下矩阵的右除运算还可以由函数 `mldivide(A,B)` 得出。

• **矩阵的右除**

MATLAB 中定义了 "/" 符号，用于表示两个矩阵的右除，相当于求方程 $\boldsymbol{XA} = \boldsymbol{B}$ 的解。$\boldsymbol{A}$ 为非奇异方阵时 `B/A` 为 $\boldsymbol{BA}^{-1}$，但在计算方法上存在差异，更精确地，有 `B/A=(A'\B')'`。MATLAB 下矩阵的右除运算还可以由函数 `mrdivide(A,B)` 得出。

• **矩阵翻转**

MATLAB 提供了一些矩阵翻转处理的特殊命令，如 `B=fliplr(A)` 命令将矩阵 $\boldsymbol{A}$ 进行左右翻转再赋给 $\boldsymbol{B}$，亦即 $b_{ij} = a_{i,n+1-j}$，而 `C=flipud(A)` 命令将 $\boldsymbol{A}$ 矩阵进行上下翻转并将结果赋给 $\boldsymbol{C}$，亦即 $c_{ij} = a_{m+1-i,j}$。`D=rot90(A)` 将 $\boldsymbol{A}$ 矩阵逆时针旋转 90° 后赋给 $\boldsymbol{D}$，亦即 $d_{ij} = a_{j,n+1-i}$。用户可以通过下面的例子来体会各个函数的翻转效果。

〖例 2.3〗 假设 A 矩阵为 A=[1,2,3; 4,5,6; 7,8,0]，调用这三个函数分别得出如下的结果：

```
>> A=[1,2,3; 4,5,6; 7,8,0]  % 输入 A 矩阵
A =
     1     2     3
     4     5     6
     7     8     0
>> B=fliplr(A) % 左右翻转
B =
     3     2     1
     6     5     4
     0     8     7
>> C=flipud(A) % 上下翻转
C =
     7     8     0
     4     5     6
     1     2     3
>> D=rot90(A)  % 旋转 90 度
D =
     3     6     0
     2     5     8
     1     4     7
```

- **矩阵乘方运算**

一个矩阵的乘方运算可以在数学上表述成 A^x，而其前提条件要求 A 矩阵为方阵。如果 x 为正整数，则乘方表达式 A^x 的结果可以将 A 矩阵自乘 x 次得出。如果 x 为负整数，则可以将 A 矩阵自乘 $-x$ 次，然后对结果进行求逆运算就可以得出该乘方结果。如果 x 是一个分数，例如 $x = n/m$，其中 n 和 m 均为整数，则首先应该将 A 矩阵自乘 n 次，然后对结果再开 m 次方。

矩阵的开方运算是相当困难的，但有了数字计算机，这种运算就不再显得那么麻烦了，用户可以利用计算机方便地求出一个矩阵的方根。

在 MATLAB 环境下，如果给定了一个 A 矩阵，则其乘方矩阵和开方矩阵可以容易地由 `A^x` 求出，其中 `x` 为一个常数。矩阵的乘方运算还可以写成 `mpower(A,x)`。

〖例 2.4〗 仍考虑前面给出的 A 矩阵

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

该矩阵的平方和 0.1 次幂可以由 `A^2` 和 `A^0.1` 两条命令容易地求得。

```
>> A^2 % 矩阵平方
ans =
```

```
    30    36    15
    66    81    42
    39    54    69
>> A^0.1 % A 阵开 10 次方，得出 10 个矩阵中的主矩阵
ans =
   0.9750 + 0.2452i   0.1254 - 0.0493i   0.0059 - 0.0604i
   0.2227 - 0.0965i   1.1276 + 0.1539i   0.0678 - 0.1249i
   0.0324 - 0.1423i   0.0811 - 0.1659i   1.1786 + 0.2500i
```

事实上，$\boldsymbol{A}$ 矩阵开 10 次方应该有 10 个解，用 MATLAB 的乘方运算只能得出其中一个根。

- **点运算**

MATLAB 中定义了一种特殊的运算，即所谓的点运算。两个矩阵之间的点运算是它们对应元素的直接运算。例如 `C=A.*B` 表示 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的相应元素之间直接进行乘法运算，然后将结果赋给 $\boldsymbol{C}$ 矩阵，即，$c_{ij} = a_{ij}b_{ij}$。这种点乘积运算又称为 Hadamard 乘积。注意，点乘积运算要求 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的维数相同。可以看出，这种运算和普通乘法运算是不同的。

〖例 2.5〗 对给定的两个简单矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \\ 8 & 9 & 0 \end{bmatrix}$$

由下面的 MATLAB 语句则可以得出如下结果：

```
>> A=[1 2 3; 4 5 6; 7 8 0]; B=[2 3 4; 5 6 7; 8 9 0];
   A*B  % 矩阵乘积
ans =
    36    42    18
    81    96    51
    54    69    84
>> A.*B  % 矩阵点乘
ans =
     2     6    12
    20    30    42
    56    72     0
```

可以看出，这两种乘积结果是不同的，前者是普通矩阵乘积，而后者是两个矩阵对应元素之间的乘积，所以采用点运算时要注意其含义。

点运算在 MATLAB 中起着很重要的作用，例如当 x 是一个向量时，则求取数值 x^5 时不能直接写成 `x^5`，而必须写成 `x.^5`。在进行矩阵的点运算时，同样要求运算的两个矩阵的维数一致，或其中一个变量为标量。其实一些特殊的函数，如 `sin()` 也是由点运算的形式进行的，因为它要对矩阵的每个元素求取正弦值。矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的点乘积运算可以由 `times(A,B)` 来实现。

矩阵点运算不光可以用于点乘积运算，还可以用于其他运算的场合。比如对前面给出的 A 矩阵作 A.^A 运算，则将得出下面的结果：

```
>> A.^A
ans =
          1           4          27
        256        3125       46656
     823543    16777216           1
```

在这一例子中，每个矩阵元素都是原来元素的相应乘方，例如其中的数据$823543=7^7$。点乘方运算的 MATLAB 表示还可以写成 power(A,B)。

- **Kronecker 乘积**

若存在两个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$，其中 $\boldsymbol{A}$ 为 $n\times m$ 阶矩阵，$\boldsymbol{B}$ 为 $p\times q$ 阶矩阵，则 $\boldsymbol{A}$ 与 $\boldsymbol{B}$ 矩阵的 Kronecker 乘积运算可以定义为

$$\boldsymbol{C}=\boldsymbol{A}\otimes\boldsymbol{B}=\begin{bmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \cdots & a_{1m}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} & \cdots & a_{2m}\boldsymbol{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}\boldsymbol{B} & a_{n2}\boldsymbol{B} & \cdots & a_{nm}\boldsymbol{B} \end{bmatrix} \tag{2.2}$$

由上面式子可以看出，Kronecker 乘积 $\boldsymbol{A}\otimes\boldsymbol{B}$ 与 $\boldsymbol{B}\otimes\boldsymbol{A}$ 均为 $np\times mq$ 阶矩阵，但一般情况下 $\boldsymbol{A}\otimes\boldsymbol{B}\neq\boldsymbol{B}\otimes\boldsymbol{A}$。和普通矩阵乘积不同，Kronecker 乘积并不要求两个被乘的矩阵满足任何意义下的维数匹配。Kronecker 积的 MATLAB 命令为 C=kron(A,B)。

〖例 2.6〗 例如给定两个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$

$$\boldsymbol{A}=\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix} 1 & 3 & 2 \\ 2 & 4 & 6 \end{bmatrix}$$

则 $\boldsymbol{A}$ 与 $\boldsymbol{B}$ 的 Kronecker 积可以由下面的 MATLAB 命令求出来。

```
>> A = [1, 2;  3, 4];  B = [1, 3, 2; 2, 4, 6];
   C = kron(A, B)
C =
     1     3     2     2     6     4
     2     4     6     4     8    12
     3     9     6     4    12     8
     6    12    18     8    16    24
>> D=kron(B,A)
D =
     1     2     3     6     2     4
     3     4     9    12     6     8
     2     4     4     8     6    12
     6     8    12    16    18    24
```

2.3.2 矩阵的逻辑运算

MATLAB 语言并没有定义专门的逻辑变量，在 MATLAB 语言中，如果一个数的值为 0，则可以认为它为逻辑 0，否则为逻辑 1。

假设矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 均为 $n \times m$ 矩阵，则在 MATLAB 下定义了下面的逻辑运算：

- **矩阵的与运算**

在 MATLAB 下用 `&` 号表示矩阵的与运算，例如 `A & B` 表示两个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的与。MATLAB 中定义的与运算应该注意下面几个问题：

(1) $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的维数应该相同或其中之一为标量，否则不能进行与运算。

(2) $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的对应元素进行“与”运算。在 MATLAB 定义下，如果两个数均非 0 则该结果元素的值为 1，否则该元素为 0。

(3) 矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的与运算还可以表示成 `and(A,B)`。

- **矩阵的或运算**

在 MATLAB 下用 `|` 号表示矩阵的或运算，例如 `A | B` 表示两个矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的或。MATLAB 中定义的或运算应该注意下面几个问题：

(1) $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的维数应该相同或其中之一为标量，否则不能进行或运算。

(2) $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的对应元素进行“或”运算。在 MATLAB 定义下，如果两个数均为 0 则该结果元素的值为 0，否则该元素为 1。

(3) 矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的或运算还可以表示成 `or(A,B)`。

- **矩阵的非运算**

在 MATLAB 下用 `~` 号表示矩阵的非运算。例如 `~A` 表示 $\boldsymbol{A}$ 矩阵的非运算。MATLAB 中定义的非运算应该注意下面几个问题：

(1) $\boldsymbol{A}$ 矩阵进行“非”运算时，若矩阵对应元素为 0，则结果为 1，否则为 0。

(2) 矩阵 $\boldsymbol{A}$ 非运算还可以表示成 `not(A)`。

- **矩阵的异或运算**

MATLAB 下矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的异或运算可以表示成 `xor(A,B)`。MATLAB 中定义的异或运算应该注意下面几个问题：

(1) $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的维数应该相同或其中之一为标量，否则不能进行异或运算。

(2) $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 矩阵的对应元素进行“异或”运算。在 MATLAB 定义下，若相应的两个数一个为 0，一个非 0，则结果为 0，否则为 1。

【例 2.7】 假设两个矩阵分别为

$$\boldsymbol{A}=\begin{bmatrix}0 & 2 & 3 & 4\\ 1 & 3 & 5 & 0\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}1 & 0 & 5 & 3\\ 1 & 5 & 0 & 5\end{bmatrix}$$

可以由下面的 MATLAB 语句对它们进行各种逻辑运算。

```
>> A=[0 2 3 4;1 3 5 0]; B=[1 0 5 3;1 5 0 5];
   A & B                    % 与运算
ans =
     0     0     1     1
```

```
     1     1     0     0
>> A | B                 % 或运算
ans =
     1     1     1     1
     1     1     1     1
>> ~A                    % 非运算
ans =
     1     0     0     0
     0     0     0     1
>> xor(A,B)              % 异或运算
ans =
     1     1     0     0
     0     0     1     1
```

2.3.3 矩阵的比较关系

MATLAB 语言定义了各种比较关系，如：

>	大于关系	<	小于关系
==	等于关系	>=	大于或等于关系
<=	小于或等于关系	~=	不等于关系

这些关系运算都是针对两个矩阵对应元素的，所以在使用关系运算时，首先应该保证两个矩阵的维数是一致的或其一为标量。关系运算对两个矩阵的对应运算进行比较，若关系满足，则将结果矩阵中该位置的元素置为 1，不满足则置 0。

〖例 2.8〗 对上面例子中两个矩阵进行关系运算，则可以得出如下结果。

```
>> A=[0 2 3 4;1 3 5 0]; B=[1 0 5 3;1 5 0 5];
   A == B      % 判定相等关系
ans =
     0     0     0     0
     1     0     0     0
>> A >= B      % 判定大于、等于关系
ans =
     0     1     0     1
     1     0     1     0
>> C=B~=A      % 判定不等于关系
C =
     1     1     1     1
     0     1     1     1
```

MATLAB 还提供了一些特殊的函数，在编程中也是很实用的，其中 find() 函数可

以查询出满足某关系的数组下标，例如，若想查出矩阵 C 中数值等于 1 的元素下标，则可以给出 find(C==1) 命令，

```
>> find(C==1)
ans =
     1
     3
     4
     5
     6
     7
     8
```

可以看出，该函数相当于先将 C 矩阵先按列构成列向量，然后再判断哪些元素为 1，返回其下标。而 **find(isnan(A))** 函数将查出 **A** 变量中为 **NaN** 的各元素下标。还可以用下面的格式同时返回行和列坐标

```
>> [i,j]=find(C==1); [i,j]
ans =
     1     1
     1     2
     2     2
     1     3
     2     3
     1     4
     2     4
```

此外，前面介绍的 all() 和 any() 函数也是很实用的查询函数，其中 all(C==1) 和 any(C==1) 将得出如下结果

```
>> all(C==1)
ans =
     0     1     1     1
>> any(C==1)
ans =
     1     1     1     1
```

前一个命令当 C 矩阵的某列元素全非零时相应的结果元素为 1，否则为 0; 而后者在某列中含有非零元素时得出的相应元素为 1，否则为 0。对向量 x 进行操作时，all(x) 在 x 中所有元素均非 0 时返回 1，否则为 0。而 any(x) 在 x 中有非 0 元素时返回 1，否则为 0。另外，all() 和 any() 函数还可以相互嵌套，例如若想判定一个矩阵 C 是否元素均非零，则可以使用 all(all(C))，更简单地 all(C(:))。

2.3.4 矩阵元素的数据变换

对由小数构成的矩阵 A 来说，如果想对它取整数，则有如下几种方案：

(1) floor(A) 将 A 中元素按 $-\infty$ 方向取整，即取不足整数；

(2) ceil(A) 将 A 中元素按 $+\infty$ 方向取整，即取过剩整数；

(3) round(A) 将 A 中元素按最近的整数取整，亦即四舍五入；

(4) fix(A) 将 A 中元素按离 0 近的方向取整。

〖例 2.9〗 考虑如下生成的随机数矩阵 A

```
>> A=-2.5+5*rand(3)  % 生成 [-2.5,2.5] 区间上的均匀分布随机数
A =
   -0.2765    2.1091   -0.4715
    0.5772    1.1910    2.1773
    1.4597   -1.6187    2.0845
```

结合上面 4 个函数的结果，可以更好地理解各个函数的作用。

```
>> floor(A)  % 向 -Inf 方向取整
ans =
    -1     2    -1
     0     1     2
     1    -2     2
>> ceil(A)   % 向 +Inf 方向取整
ans =
     0     3     0
     1     2     3
     2    -1     3
>> round(A)  % 向 0 的方向取整
ans =
     0     2     0
     1     1     2
     1    -2     2
>> fix(A)   % 取最近的整数
ans =
     0     2     0
     0     1     2
     1    -1     2
```

此外，一些小数还可以用两个整数的除式形式表示成有理数形式，MATLAB 中提供了 rat() 函数，用它可以获得矩阵的有理数近似。该函数的调用格式为 [n,d]=rat(A)，其中，A 中的每个元素均可以表示成为两个整数的有理除式，亦即 A=n./d。

〖例 2.10〗 考虑 4 阶 Hilbert 矩阵，可以用 rat() 函数求出各个元素的有理表示

```
>> A=hilb(4)
A =
    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
>> [n,d]=rat(A)
n =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
d =
     1     2     3     4
     2     3     4     5
     3     4     5     6
     4     5     6     7
```

MATLAB 还提供了 rem() 函数，可以求取矩阵元素的余数，该函数的调用格式为 C=rem(A,x)，表示 A 矩阵除以模数 x 后的余数。若 $x\neq 0$，则整数部分由 fix(A./x) 表示，余数 C=A-x.*fix(A./x)。如果 $x=0$，则定义 rem(A,0)=NaN，MATLAB 允许模数 x 为小数。

重新考虑上面的例子，可以求出 A 矩阵在模 0.3 下的余数为：

```
>> rem(A,0.3)
ans =
    0.1000    0.2000    0.0333    0.2500
    0.2000    0.0333    0.2500    0.2000
    0.0333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
```

值得指出的是，上面叙述的各个函数均不影响 NaN 和 Inf 等常量。更一般地，x 可以是和 A 相同大小的矩阵，这时将按照对应元素进行计算。

2.4　流程控制结构

作为一种程序设计语言，MATLAB 提供了循环语句结构、条件语句结构、开关语句结构以及新的试探语句。本节中将介绍各种语句结构。

2.4.1　循环结构

循环语句有两种结构：for ... end 结构和 while ... end 结构。这两种语句结构不

完全相同，各有各的特色。for ... end 语句通常的调用格式为：

```
for 循环变量=Vect
   循环体语句组
end
```

其中 Vect 为任意给定的向量，该语句的作用即循环变量从 Vect 向量中的第 1 个数值一直循环到最后一个数值，和前面的 for 格式不同，在这种调用格式下并不要求循环变量作等距选择，也不要求它是单调的，所以在一些特殊的情况下，这种调用格式是很实用的。

注意，这里的循环语句是以 end 结尾的，这和 C 语言的结构不完全一致。在 C 语言循环中，循环体的内容是以大括号 {} 括起来的，而在 MATLAB 语言中，循环体的内容是以循环语句和 end 语句括起来的，所以在使用 MATLAB 时应注意这一点。

通常使用的循环格式为 for i=s1:s3:s2，其循环体结构的程序框图表示在图 2-5 (a) 中给出，如果 s3>0，其效果和 C 语言中的 for (i=s1; i<=s2; i+=s3) 是一致的，而在 s3<0 时它和 for (i=s1; i>=s2; i+=s3) 是一致的。

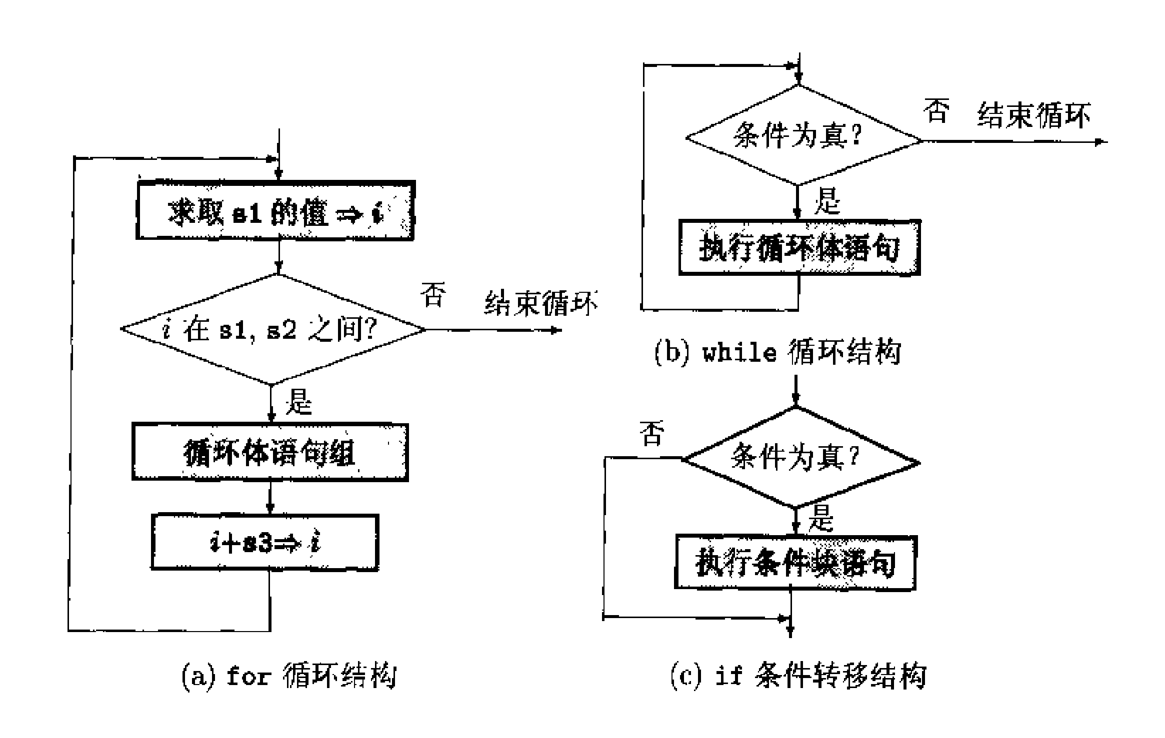


图 2-5 MATLAB 下各种程序结构框图

在 MATLAB 的循环语句基本格式中，循环变量可以取作任何 MATLAB 变量。表达式 s1, s2 和 s3 的定义和 C 语言相似，即首先将循环变量的初值赋成 s1 的值，然后再求取 s2 的值，如果此时循环变量的值介于 s1 和 s2 的值之间，则执行循环体中的语句，否则结束循环语句的执行。执行完一次循环体中的语句之后，则会将循环变量自增一个 s3 的值，然后再判断循环变量是否介于 s1 和 s2 之间，如果满足仍再执行循环体，直至不

满足为止。这时将结束循环语句的执行，而继续执行后面的语句。

〖例 2.11〗 如果用户想由 MATLAB 求出 $\sum_{i=1}^{100} i$ 的值，可以作下列的循环:

```
>> mysum=0;
   for i=1:1:100,
      mysum=mysum+i;
   end;
   mysum
mysum =
        5050
```

在上面的式子中，可以看到 for 循环语句中 s3 的值为 1。在 MATLAB 实际编程中，如果 s3 的值为 1，则可以在该语句中省略，故该语句可以简化成 for i=1:100。

在实际编程中，在 MATLAB 下采用循环语句会降低其执行速度，所以前面的程序可以由下面的命令来代替: i=1:100; mysum=sum(i)。在这一语句中，首先生成了一个向量 i，然后用内部函数 sum() 求出 i 向量的各个元素之和，或更简单地，该语句还可以写成 sum(1:100)。如果前面的 100 改成 10000，再运行这一程序，则可以明显地看出，后一种方法编写的程序比前一种方法快得多。

〖例 2.12〗 假设控制系统的数学模型为 $(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{D})$，在 MATLAB 下可以由下面语句输入

```
>> A=[0 1; -1 -2]; B=[0;1]; C=[1 2]; D=0;
```

已知在频率点 ω 处，频域响应的值可以由下式求出

$$\boldsymbol{M}(\omega)=\boldsymbol{C}(\mathrm{j}\omega\boldsymbol{I}-\boldsymbol{A})^{-1}\boldsymbol{B}+\boldsymbol{D}$$

其中 $\mathrm{j}=\sqrt{-1}$, $\boldsymbol{I}$ 为单位矩阵。通常对频率没有必要均匀取值，而应该针对不同的频率段单独取值。例如在 0.1 到 1000 的范围内每个 10 倍频取十几个点，这样就可以得出:

```
>> w=[0.1:0.05:1, 1.5:0.5:10, 15:5:100 150:50:1000];
   m=[]; I=eye(size(A));
   for i=w,
      m=[m; C*inv(sqrt(-1)*i*I-A)*B+D];
   end
   length(w)   % 显示频率点个数
ans =
     73
```

可见，这样只需 73 个频率点即可以获得平滑的曲线，若从 0.1 到 1000 进行等间距选点、并假定选择间距为 0.1，可以给出下面的 MATLAB 语句

```
>> w1=0.1:.1:1000; m1=[];
   for i=w1,
      m1=[m1; C*inv(sqrt(-1)*i*I-A)*B+D];
   end
   length(m1)
```

```
ans =
    10000
```

这固然可以保证曲线的平滑，但需要选择 10000 个频率点。而选择间距为 1

```
>> w2=0.1:1:1000; m2=[];
   for i=w2,
      m2=[m2; C*inv(sqrt(-1)*i*I-A)*B+D];
   end
   length(m2)
ans =
    1000
```

则也需要 1000 个频率点，但低频处频率点太稀，而高频处又太密，甚至密得没必要，所以这样的选择也不是很理想。

MATLAB 语言提供了另一种循环语句结构 **while ... end** 语句，该循环结构的基本格式为：

```
while 逻辑变量
   循环体语句组
end
```

该循环结构的执行方式为：若逻辑变量为真(或 1，其实在 MATLAB 语言中更确切地说是"非 0")，则执行循环体的内容，执行后再返回 **while** 引导的语句处，判断该逻辑变量是否仍然为真，如果为 0 则跳出循环，向下继续执行。**while** 循环结构的框图表示如图 2-5(b) 所示。

重新考虑例 2.11，如果改用 **while** 循环结构，则可以写出下面的程序：

```
>> mysum = 0; i=1;
   while (i<=100),
      mysum=mysum+i; i=i+1;
   end
```

当然，MATLAB 提供的循环结构 **for** 和 **while** 是允许多级嵌套的，而且它们之间也允许相互嵌套，这和 C 语言等高级程序设计语言是一致的。

【例 2.13】 如果将例 2.11 中给出的问题变成求出满足 $\sum_{i=1}^{m} i > 10000$ 的最小 m 值，这样就不能直接调用 sum() 函数了。用户可以针对这一问题编写如下的程序段

```
>> mysum=0; i=1;
   while mysum<=10000,
      mysum=mysum+i; i=i+1;
   end
   [m, mysum]
ans =
```

```
   142      10011
```

与循环语句相关的还有一个重要的 break 语句，当在循环体内执行到该语句时，则程序将无条件地跳出本层循环。该语句的使用将结合后面的条件转移语句来介绍。

2.4.2 条件转移结构

除了前面介绍的循环语句结构之外，MATLAB 还提供了各种条件转移语句的结构，使得 MATLAB 语言更易于使用。MATLAB 提供的条件语句最简单的格式是由关键词 if 引导的，其格式为：

```
if 逻辑变量
   条件块语句组
end
```

其结构的框图如图 2-5 (c) 所示，当给出的逻辑变量为非 0 时，则执行该条件块结构中的语句组内容，执行完之后继续向下执行；若该逻辑变量为 0，则跳过条件块语句组而直接向下执行。

【例 2.14】 再考虑例 2.13 中给出的问题，用户可以针对这一问题编写如下的程序段

```
>> mysum = 0;
   for m=1: 1000,
      if (mysum>10000), break; end
      mysum = mysum+m;
   end
```

这时得出的结果和前面完全一致。注意，这里使用了 break 命令，其作用就是中止上一级的 for 或 while 语句引导的循环过程。

当然，前面介绍的 if 条件结构只能处理较简单的条件，所以其功能不是很全面。MATLAB 还提供了其他两种条件结构，if ... else 格式和 if ... elseif ... else 格式，这两种格式的调用方法分别为：

```
if 条件式
  条件块语句组1
else
  条件块语句组2
end
```

```
if 条件式1
  条件块语句组1
elseif 条件式2
  条件块语句组2
  ⋮
else
  条件块语句组n+1
end
```

其框图表示分别如图 2-6(a) 和 (b) 所示。可见这些语句的结构和功能与其他的程序设计

语言 (如 C 和 FORTRAN) 是基本一致的。后面将通过例子来说明这样条件结构的使用方法。

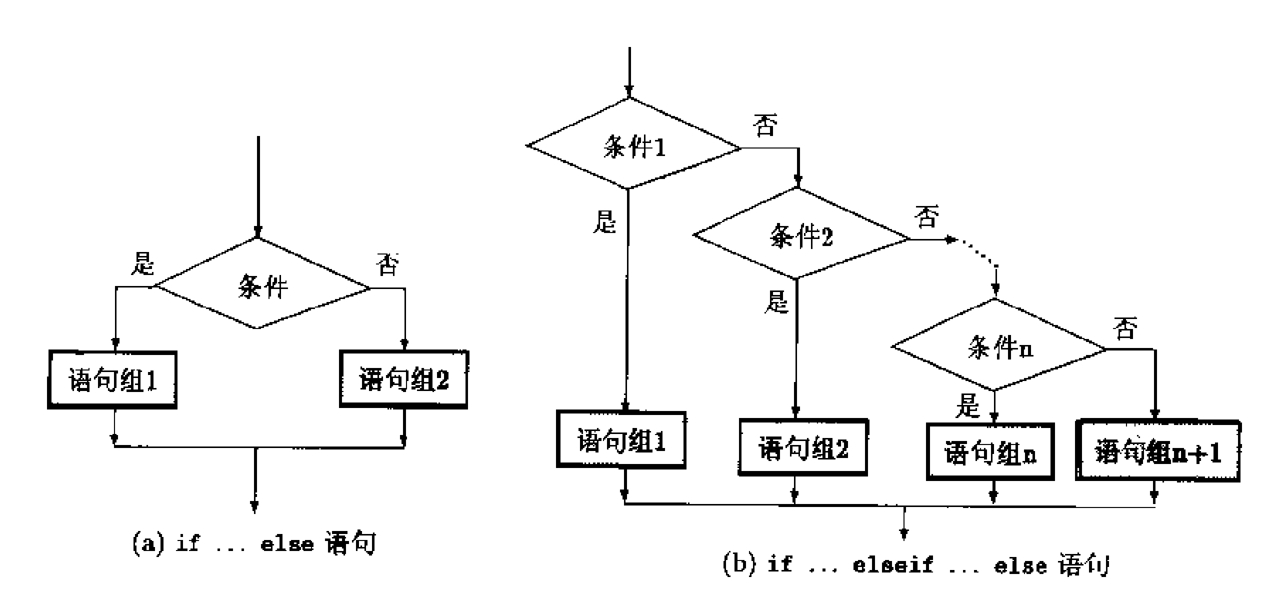


图 2-6 条件语句结构框图

2.4.3 开关结构

MATLAB 从 5.0 版开始提供了开关语句结构，其基本语句结构为：

```
switch 开关表达式
case 表达式1
  语句段1
case {表达式2,表达式3,…, 表达式m}
  语句段2
    ⋮
otherwise
  语句段n
end
```

开关语句的基本结构如图 2-7 所示。开关语句的关键是对开关表达式值的判断，当开关表达式的值等于某个 case 语句后面的条件时，程序将转移到该组语句中执行，执行完成后程序转出开关体继续向下执行。在使用开关语句结构时应该注意下面几点：

- 当开关表达式的值等于表达式 1 时，将执行语句段 1，执行完语句段 1 后将转出开关体，无需像 C 语言那样在下一个 case 语句前加 break 语句，所以本结构在这点上和 C 语言是不同的。
- 当需要在开关表达式满足若干个表达式之一时执行某一程序段，则应该把这样的一些

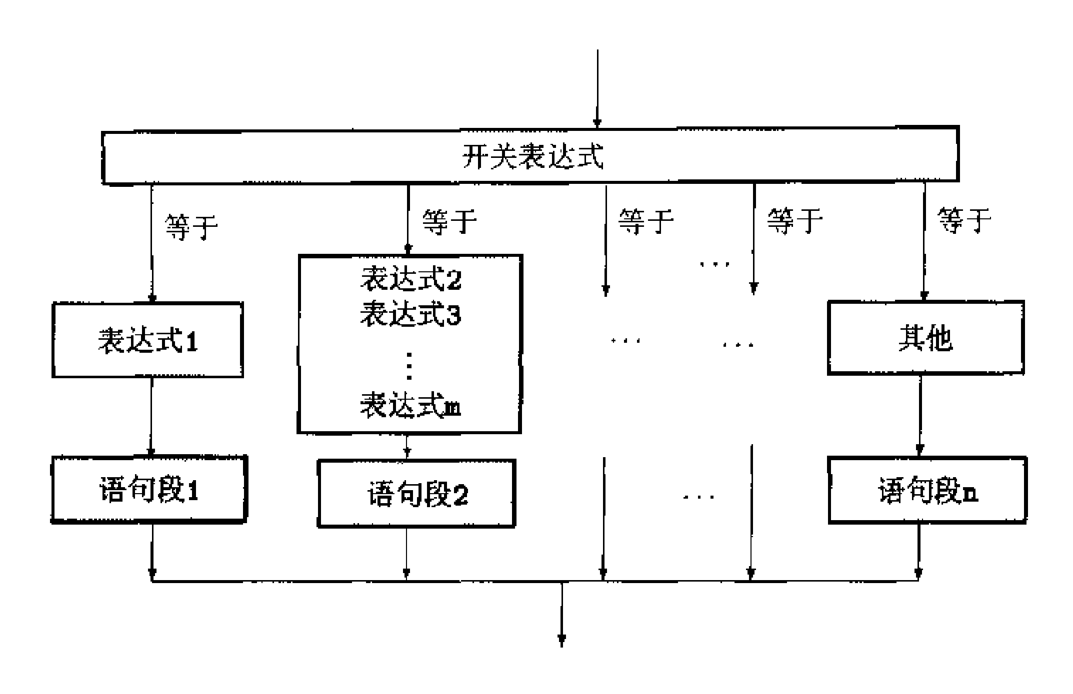


图2-7　开关语句结构框图

表达式用大括号括起来，中间用逗号分隔。事实上，这样的结构是MATLAB语言定义的单元结构。

- 当前面枚举的各个表达式均不满足时，则将执行 otherwise 语句后面的语句段，此语句等价于C语言中的 default 语句。
- 程序的执行结果和各个 case 语句的次序是无关的。当然这也不是绝对的，当两个 case 语句中包含同样的条件，执行结果则和这两个语句的顺序有关。
- 在 case 语句引导的各个表达式中，不要用重复的表达式，否则列在后面的开关通路将永远也不能执行。

2.4.4　试探式语句结构

MATLAB从5.2版本开始提供了一种新的试探式语句结构，其一般的形式为：



```
try
  语句段1
catch
  语句段2
end
```

本语句结构首先试探性地执行语句段1，如果在此段语句执行过程中出现错误，则将错误信息赋给保留的 lasterr 变量，并放弃这段语句，转而执行语句段2中的语句。这种新的语句结构是C等语言中所没有的。

试探性结构在实际编程中还是很实用的，例如可以将一段不保险但速度快的算法放到 try 段落中，而将一个保险的程序放到 catch 段落中，这样就能保证原始问题的求解更加可靠，且可能使程序高速执行。

2.5 MATLAB 函数编写与技巧

MATLAB 下提供了两种源程序文件格式，其中一种是普通的ASCII码构成的文件，在这样的文件中包含一族由 MATLAB 语言所支持的语句，它类似于 DOS 下的批处理文件，这种文件称作 M 脚本文件 (M-script，本书中将其简称为 M 文件)，它的执行方式很简单，用户只需在 MATLAB 的提示符 >> 下键入该 M 文件的文件名，这样 MATLAB 就会自动执行该 M 文件中的各条语句。M 文件只能对 MATLAB 工作空间中的数据进行处理，文件中所有语句的执行结果也完全返回到工作空间中。M 文件格式适用于用户所需要立即得到结果的小规模运算。

例如，为例 2.12 编写的MATLAB 源程序就是一个M 文件，因为它直接对 MATLAB 工作空间中的变量进行访问。如果系统模型的 A, B, C, D 变量换了变量名，则此文件就不能实现上述的计算了。

另一种源程序格式是 M 函数格式，它是 MATLAB 程序设计的主流，一般情况下，不建议使用 M 脚本文件格式编程。本节将着重介绍 MATLAB 函数的编写方法与技巧。

2.5.1 MATLAB 语言的函数的基本结构

MATLAB 的 M 函数是由 function 语句引导的，其基本结构如下：

```
function [返回变量列表] = 函数名 (输入变量列表)
注释说明语句段，由 % 引导
输入、返回变量格式的检测
函数体语句
```

这里输入和返回变量的实际个数分别由 nargin 和 nargout 两个 MATLAB 保留变量来给出，只要进入该函数，MATLAB 就将自动生成这两个变量。

返回变量如果多于 1 个，则应该用方括号将它们括起来，否则可以省去方括号。输入变量和返回变量之间用逗号来分割。注释语句段的每行语句都应该由百分号 (%) 引导，百分号后面的内容不执行，只起注释作用。用户采用 help 命令则可以显示出来注释语句段的内容。此外，正规的变量个数检测也是必要的。如果输入或返回变量格式不正确，则应该给出相应的提示。这里将通过下面的例子来演示函数编程的格式与方法。

〖例 2.15〗 假设想生成一个 $n\times m$ 阶的 Hilbert 矩阵[①]，它的第 i 行第 j 列的元素值为 $1/(i+j-1)$。

[①] MATLAB 中提供了生成 Hilbert 矩阵的函数 hilb()，这里只是演示函数的编写方法，而在实际使用时还是应该采用 hilb() 函数。事实上，hilb() 函数并不能生成长方 Hilbert 矩阵。

想在编写的函数中实现下面几点:

(1) 如果只给出一个输入参数，则会自动生成一个方阵，即令 $m=n$;

(2) 在函数中给出合适的帮助信息，包括基本功能、调用方式和参数说明;

(3) 检测输入和返回变量的个数，如果有错误则给出错误信息;

(4) 如果调用时不要求返回变量，则将显示结果矩阵。

其实在编写程序时养成一个好的习惯，无论对程序设计者还是对程序的维护者、使用者都是大有裨益的。

根据上面的要求，可以编写一个 MATLAB 函数 myhilb()，文件名为 myhilb.m，并应该放到 MATLAB 的路径下。

```
function A=myhilb(n, m)
%MYHILB  本函数用来演示 MATLAB 语言的函数编写方法。
%   A=MYHILB(N, M) 将产生一个 N 行 M 列的 Hilbert 矩阵 A;
%   A=MYHILB(N) 将产生一个 NxN 的方 Hilbert 阵 A;
%   MYHILB(N,M) 调用格式只显示 NxM 的 Hilbert 矩阵，但不返回任何矩阵。
%
%See also: HILB.

%  Designed by Professor Dingyu XUE, Northeastern University, PRC
%     5 April, 1995, Last modified by DYX at 30 July, 2001
if nargout>1, error('Too many output arguments.'); end
if nargin==1, m=n;
elseif nargin==0 | nargin>2
   error('Wrong number of iutput arguments.');
end
A1=zeros(n,m);
for i=1: n
   for j=1:m
     A1(i,j)=1/(i+j-1);
end, end
if nargout==1, A=A1; elseif nargout==0, disp(A1); end
```

在这段程序中，由 % 引导的部分是注释语句，通常用来给出一段说明性的文字来解释程序段落的功能和变量含义等。由前面的第 (1) 点要求，首先测试输入的参数个数，如果个数为 1 (即 nargin 的值为 1)，则将矩阵的列数 m 赋成 n 的值，从而产生一个方阵。如果输入或返回变量个数不正确，则函数前面的语句将自动检测，并显示出错误信息。后面的双重 for 循环语句依据前面给出算法来生成一个 Hilbert 矩阵。最后根据前面的第 (4) 点要求，判断返回参数的个数，如果个数为 0，则只显示生成的矩阵，而不向主调函数返回任何信息。

此函数的联机帮助信息可以由下面的命令获得

```
>> help myhilb
```

```
MYHILB  本函数用来演示 MATLAB 语言的函数编写方法。
    A=MYHILB(N, M) 将产生一个 N 行 M 列的 Hilbert 矩阵 A;
    A=MYHILB(N) 将产生一个 NxN 的方 Hilbert 阵 A
    MYHILB(N,M) 调用格式只显示 NxM 的 Hilbert 矩阵，但不返回任何矩阵。

See also: HILB.
```

注意，这里只显示了程序及调用方法，而没有把该函数中有关作者的信息显示出来。对照前面的函数可以立即发现，因为在作者信息的前面给出了一个空行，所以可以容易地得出结论：如果想使一段信息可以用 help 命令显示出来，则在它前面不应该加空行；即使想在 help 中显示一个空行，这个空行也应该由 % 来引导。

有了函数之后，可以采用下面的各种方法来调用它，并产生出所需的结果。

```
>> A=myhilb(3,4)  % 两个输入参数，返回长方形矩阵
A =
    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
>> A=myhilb(4)     % 一个输入参数，输出方阵
A =
    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
>> myhilb(4)       % 无返回参数，只显示不赋值
    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
```

〖例 2.16〗 MATLAB 函数是可以递归调用的，亦即在函数的内部可以调用函数自身。考虑求阶乘 $n!$ 的例子：由阶乘定义可见 $n! = n(n-1)!$，这样，n 的阶乘可以由 $n-1$ 的阶乘求出，而 $n-1$ 的阶乘可以由 $n-2$ 的阶乘求出，依此类推，直到计算到已知的 $1! = 0! = 1$，从而能建立起递归调用的关系（为了节省篇幅起见，这里略去了注释行段落）：

```
function k=my_fact(n)
if nargin~=1, error('输入变量个数错误，只能有一个输入变量'); end
if nargout>1, error('输出变量个数过多'); end
if abs(n-floor(n))>eps | n<0 % 判定 n 是否为非负整数
   error('n 应该为非负整数');
end
if n>1       % 如果 n>1，进行递归调用
```

```
      k=n*my_fact(n-1);
   elseif any([0 1]==n) % 0!=1!=1 为已知
      k=1;
   end
```

可以看出，该函数首先判定 n 是否为非负整数，如果不是则给出错误信息，如果是，则在 $n>1$ 时递归调用该程序自身，若 $n=1$ 或 0 时则直接返回 1。调用该函数则立即可以得出 11 的阶乘

```
>> my_fact(11)
ans =
    39916800
```

其实 MATLAB 提供了求取阶乘的函数 factorial()，其核心算法为 prod(1:n)，从结构上更简单、直观。

2.5.2 可变输入输出个数的处理

下面将介绍单元变量的一个重要应用 —— 如何建立起无限个输入或返回变量的函数调用格式。

〖例 2.17〗 MATLAB 提供的conv() 函数可以用来求两个多项式的乘积。对于多个多项式的连乘，则不能直接使用此函数，而需要用该函数嵌套使用，这样在表示很多多项式连乘时相当麻烦。在这里可以用单元数据的形式来编写一个函数 convs()，专门解决多个多项式连乘的问题。

```
function a=convs(varargin)
a=1; i=1;
while i<=length(varargin)
   k=1; a1=varargin{i}; i=i+1;
   if i<=length(varargin)
      if isa(varargin{i},'cell'), % 如果为单元数组，则表示指数
         k=varargin{i}{1}; i=i+1;
      end
   end
   for j=1:k, a=conv(a,a1); end
end
```

如果输入变量中某个变量为单元数组，则认为它是前一个多项式的指数。这时，所有的输入变量列表由单元变量 varargin 表示，相应地，如有需要，也可以将返回变量列表用一个单元变量 varargout 表示。在这样的表示下，理论上就可以处理无穷多个多项式的连乘问题了。例如可以用下面的格式调用该函数。

```
>> P=[1 2 4 0 5]; Q=[1 2]; F=[1 2 3];
   D=convs(P,Q,F)
D =
```

```
     1     6    19    36    45    44    35    30
>> T=convs([1,1],{5},[1 1]) % 亦即 (s+1)^5*(s+1)
T =
     1     6    15    20    15     6     1
>> G=convs(P,Q,F,[1,1],[1,3],[1,1])
G =
     1    11    56   176   376   578   678   648   527   315    90
```

2.5.3 MATLAB 函数的跟踪调试

MATLAB 从 4.0 版本开始就提供了程序跟踪调试的命令，5.0 版又进一步提供了跟踪调试的程序界面，MATLAB 6.x 版带有更强的跟踪调试程序，使得 MATLAB 函数的跟踪调试更加方便与直观。

函数内部的局部变量值可以由跟踪调试程序测出。下面将通过一个简单的例子来演示跟踪调试程序在 MATLAB 函数调试中的使用。考虑前面编写的 **myhilb.m** 函数，用 **edit myhilb** 命令可以打开该程序，则将得出如图 2-8 所示的编辑窗口。在断点状态栏

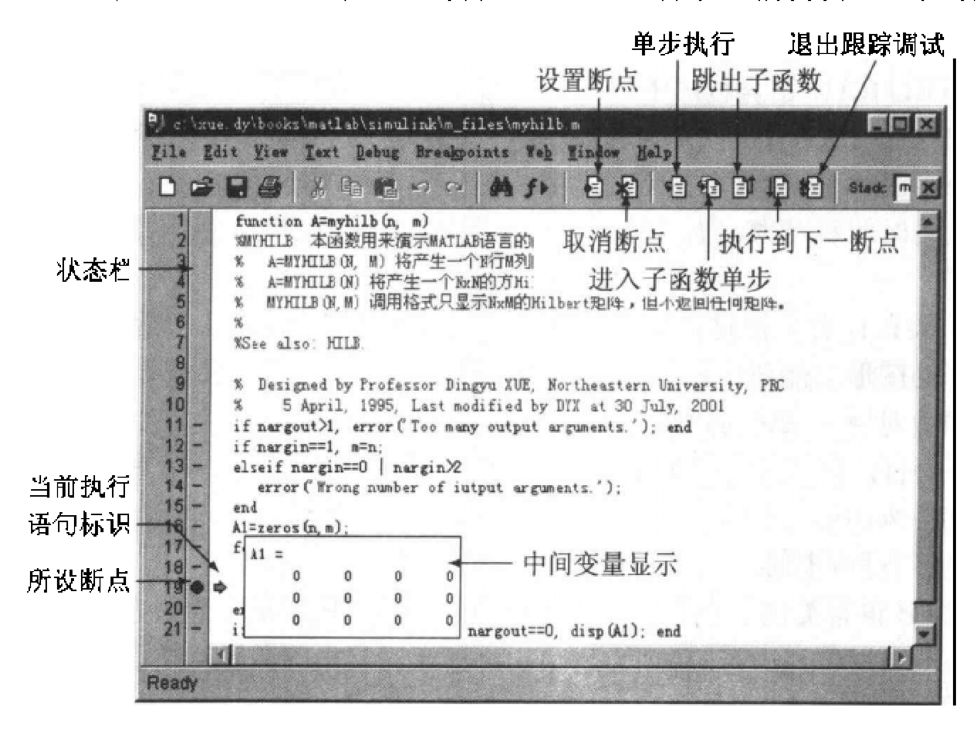


图 2-8 跟踪调试程序界面

目内单击想设置断点的位置，或在编辑窗口内将想设置断点的行设置为当前行，再单击“断点设置”按钮，则将自动地在断点状态栏目内出现红色圆点，表示断点。运行该函数到断点处，将自动中断，用户可以将鼠标移动到想查询的变量上，这时将按图 2-8 给出的方式显示该变量的内容。同时，命令窗口中的提示符将变为 K>>，用户可以在该提示符下键入函数内部变量名或表达式，这样将在命令窗口内显示其内容，并可以进行一般运

算。所以在这样的运行方式下，用户可以对函数内部的局部变量进行直接操作。

如果程序调试需要，在一个程序内可以设置多个断点。若想取消某个断点，则单击该断点对应的红点即可，如果想消除全部断点，则单击“取消断点”按钮即可。

MATLAB 的跟踪调试界面还支持“单步执行”等功能，单击该按钮则向下执行一步；若单击“进入子函数单步”按钮，则在进入子函数后仍单步执行，否则一次性执行子函数，而不进入子程序内部；若单击“跳出子函数”按钮则不再进入子函数；“执行到下一断点”按钮允许用户继续执行程序，如果后面有断点则执行到断点为止，没有断点则执行到程序结束，并退出跟踪调试的状态；“退出跟踪调试”按钮将退出跟踪调试模态，直接将程序执行到结束，命令窗口的提示符也恢复成 >>。

2.6 MATLAB 语言下图形绘制与技巧

除了 MATLAB 的强大数值分析功能外，MATLAB 语言受到工程技术人员广泛接受与使用的另一个重要原因是因为它提供了较方便的绘图功能。在 MATLAB 等软件出现之前，如果想在自己程序中生成一个图形 (即使是二维图形) 是相当困难的。例如若用户想在自己的 FORTRAN 语言程序中绘制一个图形，则首先需要对绘图的数据进行预处理，找出这些数据的最大值和最小值；然后根据它们自动地计算出坐标轴的范围；再调用一些绘图命令库函数 (例如著名的 GINO-F) 把图形在屏幕上显示出来。这样做将耗费程序设计者大量的时间和精力，而且绘制的图形效果往往还取决于设计者的经验，可能不一定令人满意。

此外如果设计者想把这样的程序移植到其他的语言下实现，例如转移成 C 语言程序，则所有的图形功能部分就必须完全地改写，这可能还要求设计者调用其他的绘图库函数，这样做对用户来讲可以说是相当苛刻的，也是一个相当沉重的负担。即使不改用其他的高级语言，若想将此程序转移到其他机器上，如从 PC 兼容机转移 Sun SPARC 工作站上，则原来的语句有许多 (尤其是绘图部分) 都需要用户重新进行改写，这样将给使用者带来很多不利的因素。

MATLAB 语言提供了强大的图形绘制功能，用户只需指定绘图方式，并提供充足的绘图数据，即可以得出所需的图形。MATLAB 还对绘出的图形提供了各种修饰方法，使绘出的图形更美观。

MATLAB 从 4.0 版本开始就提出了句柄图形学 (Handle Graphics®) 的概念，为面向对象的图形处理提供了十分有用的工具。和早期版本的 MATLAB 相比较，其最大区别在于，它在图形绘制时其中每个图形元素 (比如其坐标轴或图形上的曲线、文字等) 都是一个独立的对象。用户可以对其中任何一个图形元素进行单独地修改，而不影响图形的其他部分，具有这样特点的图形称为向量化的绘图。这种向量化的绘图要求给每个图形元素分配一个句柄 (handle)，以后再对该图形元素做进一步操作时，则只需对该句柄进行操作即可。

2.6.1 基本二维图形绘制语句

MATLAB 等新一代软件和语言使图形绘制和处理的繁杂工作变得简单得令人难以置信。如果用户将 x 和 y 轴的两组数据分别在向量 x 和 y 中存储，且它们的长度相同，则可以简单且直观地调用 **plot()** 函数。该函数最直接的调用格式为：

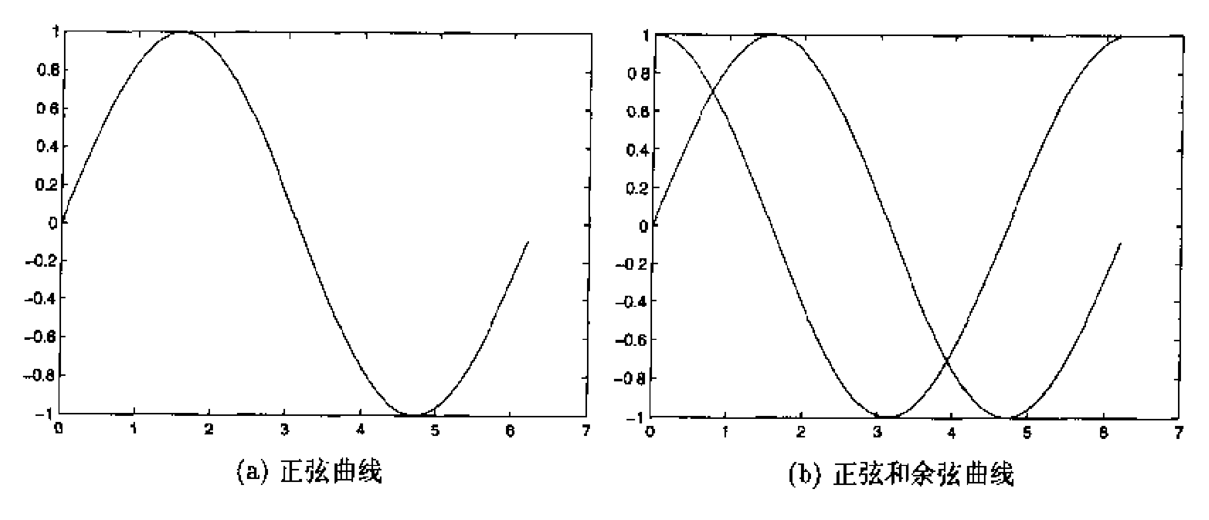
```
plot(x, y)
```

这时将在一个图形窗口上绘制出所需要的二维图形。其实，在二维图形绘制中允许用户设定许多选项，诸如是否绘制网格、x, y 坐标轴说明与图形标题设定等。调用 **plot()** 函数可以按照默认的图形格式绘制出所需曲线，至于其他选项可以通过后面将介绍的其他命令或句柄属性的修改来进一步修饰。

【例 2.18】 如果用户想绘制出一个周期内的正弦曲线，则首先应该先产生自变量 t 向量，然后由给出的自变量向量求取其正弦函数，最后调用 plot() 函数把曲线绘制出来。这个过程的 MATLAB 语言命令如下：

```
>> t=0:.1:2*pi; y=sin(t); plot(t,y)
```

上面的命令得出的结果如图 2-9(a) 所示。



(a) 正弦曲线　　(b) 正弦和余弦曲线

图 2-9　MATLAB 图形绘制举例

在 MATLAB 下还允许在一个绘图窗口上同时绘制多条曲线，例如下面的命令

```
>> t=0:.1:2*pi; y=[sin(t); cos(t)]; plot(t,y)
```

可以产生一组如图 2-9(b) 所示的曲线。这一段语句还是很好理解的，首先产生一个行向量 **t**，然后分别求取行向量 **sin(t)** 和 **cos(t)** 并将它们构成矩阵 **y** 的两行，最后将两条曲线在一个坐标系下绘制出来。从图 2-9(b) 可以容易地看出，这两条曲线都是以实线表示的，颜色深浅从黑白图上也基本分辨不出来。事实上，在彩色显示器上，MATLAB 会自动地用不同的颜色将图形显示出来。而用单色打印机打印时，也可以用不同的灰度来表示，只是有时区别不是很明显，所以出现难以辨认的情况。同理，利用这样的命令可以在图形窗口上同时绘制出多条曲线，绘制曲线的条数越多，从单色打印机输出的图上

辨认图形的困难也越大。

此外，MATLAB 还提供了 `plotyy()` 函数来绘制曲线。所不同的是，该函数绘制出来的曲线坐标轴两边均有标注。该函数的调用格式为 `plotyy(t,y1,t2,y2)`。例如，$\sin t$ 和 $0.01\cos t$ 两条曲线用此函数绘制出来的效果如图 2-10 所示。可见，此函数允许两条幅值相差悬殊的曲线在同一幅图上绘制出来，而不影响观察效果。

```
>> t=0:.1:2*pi; plotyy(t,sin(t),t,0.01*cos(t))
```

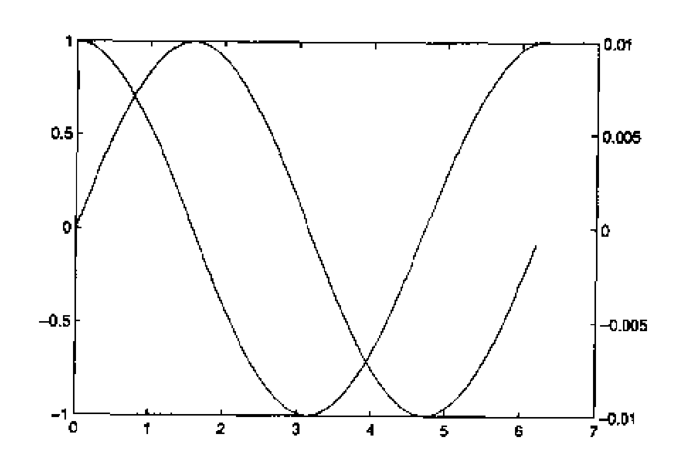


图 2-10 两边都有坐标轴的函数绘制结果

2.6.2 带有其他选项的绘图函数

MATLAB 提供了一些绘图的选项，这些选项如表 2-1 所示，其中给出的各个选项有一些可以连在一起使用，例如选项 '-.g' 表示绘制绿色的点划线。带有选项的曲线绘制

表 2-1 MATLAB 绘图命令的各种选项

曲线线型		曲线颜色				标记符号			
选项	意义	选项	意义	选项	意义	选项	意义	选项	意义
'-'	实线	'b'	蓝色	'c'	蓝绿色	'*'	星号	'pentagram'	五角星
'--'	虚线	'g'	绿色	'k'	黑色	'.'	点号	'o'	圆圈
':'	点线	'm'	红紫色	'r'	红色	'x'	叉号	'square'	□
'-.'	点划线	'w'	白色	'y'	黄色	'v'	▽	'diamond'	◇
'none'	无线	用一个 1×3 向量任意指定				'^'	△	'hexagram'	六角星
		[r,g,b] 红绿蓝三原色				'>'	▷	'<'	◁

命令的调用格式为：

```
plot(x1, y1, 选项1, x2, y2, 选项2, …)
```

该函数允许带任意多个 x 和 y 对。`plot()` 函数的调用格式是很灵活的，如果其中省

略某些选项，那么该函数也会绘制出图形。比如，下面的语句是合法的。

```
>> plot(x1, y1, 'g', x2,y2,x3,y3,':g',x4,y4)
```

除了用 x 和 y 轴的向量数据进行绘图之外，还可以给出矩阵型的数据作为 plot() 函数的参数，但同样要求 x,y 矩阵或向量参数对的维数相容，否则将给出错误信息。考虑前面用到的

```
>> t=0:.1:2*pi; y=[sin(t); cos(t)]; plot(t,y)
```

语句，其中 t 为向量，而 y 为矩阵。因为 y 矩阵的列数等于 t 向量的长度，且 y 有两行，所以该命令可以用来绘制两条曲线。

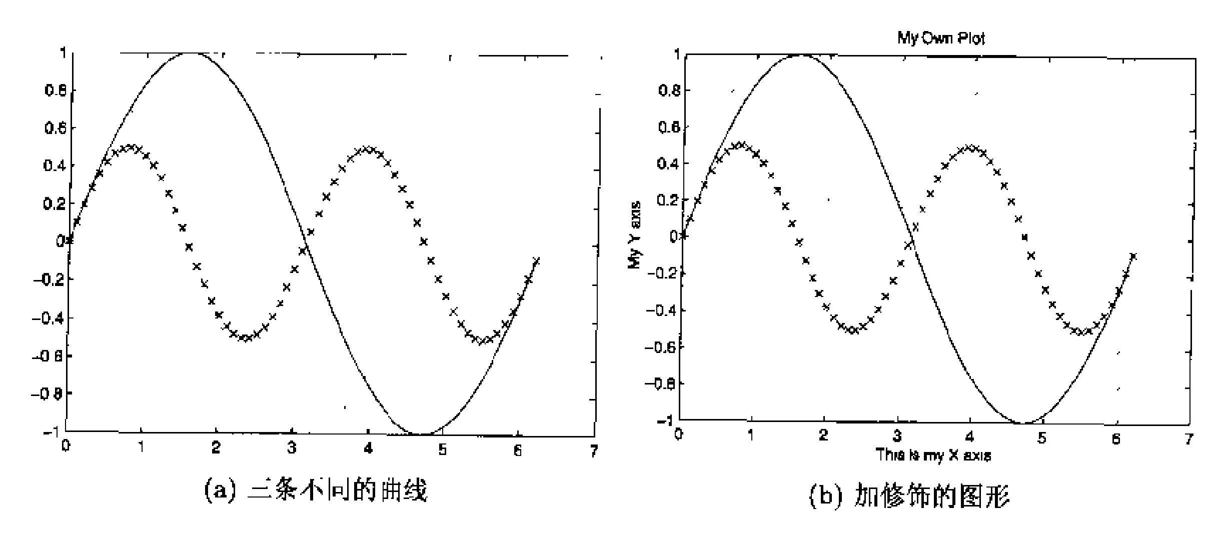
在前面的例子中，如果将语句改成

```
>> plot(t,sin(t),t,cos(t))
```

则将会得出和前面例子相同的曲线。考虑下面的命令

```
>> t=0:.1:2*pi; y1=sin(t); y2=cos(t); y3=sin(t).*cos(t);
   plot(t,y1,'-',t,y2,':',t,y3,'x')
```

分析上面的命令，可以看出总共有三条曲线，这三条曲线将分别由实线、点线和叉号来表示，如图 2-11 (a) 所示。由于绘制曲线时 x 轴上的点选择得比较密集，所以采用叉号选项时每个叉号之间的距离太近，使得它们互相重叠。在一般的实际应用中，如果使用叉号绘制图形，那么 x 轴上的点之间的距离应该选得远一点。



(a) 三条不同的曲线　　(b) 加修饰的图形

图 2-11 MATLAB 图形及其修饰

2.6.3 二维曲线的标注方法

绘制完曲线后，MATLAB 还允许用户使用它提供的一些图形修饰函数来进一步修饰画出的图形，例如若用户在绘制完上述的图形后又给出了下面的命令

```
>> grid,                          % 给图形加网格
   xlabel('This is my X axis'),   % 给 x 轴加标题
   ylabel('My Y axis'),           % 给 y 轴加标题
```

```
title('My Own Plot')            % 给图形加标题
```

则将得出如图 2-11 (b) 所示的图形显示。其中 grid 命令会自动地在各个坐标轴上加上虚线型的网格线。而 xlabel() 和 ylabel() 函数会自动地将其括号中的字符串分别写到图形的 x 轴和 y 轴附近，其中 ylabel() 函数会自动地将字符串旋转 90° 来显示。title() 函数则会将其括号中的字符串书写成该图形的标题。

和其他传统的软件不同，MATLAB 可以自动根据要绘制曲线数据的范围选择合适的坐标系，使得曲线能够尽可能清晰地显示出来。所以，在一般情况下用户不必担心绘图坐标范围的选择。但是如果用户觉得自动选择的坐标不合适，则可以用手动的方式来选择新的坐标系范围。手动变换坐标系范围的工作可以由 axis() 函数的调用来完成，该函数的调用格式为：

axis([$x_{\min}, x_{\max}, y_{\min}, y_{\max}, z_{\min}, z_{\max}$])

从这一命令可以看出，在这里用户可以自由地指定 x, y 轴，甚至 x, y, z 轴的坐标范围。如果用户只给出了四个参数，即 $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$，则系统将分别按照用户给出 x, y 轴的最小值和最大值选择坐标系范围，以便绘制出合适的二维曲线。如果除了前面的四个参数之外用户还指定了 $z_{\min}$, $z_{\max}$，则 MATLAB 在绘制三维曲线时会参照用户指定的三个坐标轴的范围来绘制出最终的图形来。

2.6.4 在 MATLAB 图形上添加文字标注

MATLAB 从 5.0 版本开始，在图形上添加各种数学公式有了更简洁的方法，在这个方法下允许用户用 TEX[52] 的格式去描述所需的文字显示。在 MATLAB 支持下的 TEX 兼容的字符串格式中，用户可以用 \bf, \it, \rm 命令分别定义黑体、斜体和正体字符，而其中一个段落可以用一对大括号 { } 括起来。例如 The {\bf word is bold} 将给出 The **word is bold** 的效果。注意 The 与 word 之前多了一个附加空格，若不想要这个空格，则应将命令改成 The {\bfword is bold}，这与标准的 TEX 命令是不完全一致的。

一些常用的符号可以由表 2-2 中给出的命令来表示。本表中的各个字符串既可以单独使用，又可以和其他字符串及命令联合使用。

除了表中给出的字符串定义之外，用户还可以通过标准的 TEX 命令格式来定义上下标，这样就可以使得图形窗口中的字符串修饰变得更丰富多采。如果想在某个字符后面加一个上标，则可以在该字符串后面跟一个 ^ 引导的字符串；若想把多个字符作为指数，则应该使用大括号。例如 y=x^{abc} 对应的显示效果为 $y = x^{abc}$，如果错误地写成 y=x^abc，则对应的排版效果为 $y = x^abc$。类似地，下标是由 _ 引导的。

此外，表示积分的算式可以由 \int_{x_0}^{x_n} 命令定义，该命令将给出 MATLAB 显示为 $\int_{x_0}^{x_n}$。在纯 TEX 下，将得出的显示效果为 $\displaystyle\int_{x_0}^{x_n}$。可见二者的显示效果不完全一致，故使用时应该注意。

【例 2.19】 用户可以用 gtext() 函数在图形窗口上加入下面的字符串，并观察修饰效果，从而进一步体会 TEX 格式的数学描述方法。

表 2-2 图形窗口下可以直接使用的 TEX 命令表

类别	c	TEX命令	c	TEX命令	c	TEX命令	c	TEX命令
小写希腊字符	α	\alpha	β	\beta	γ	\gamma	δ	\delta
	ϵ	\epsilon	ε	\varepsilon	ζ	\zeta	η	\eta
	θ	\theta	ϑ	\vartheta	ι	\iota	κ	\kappa
	λ	\lambda	μ	\mu	ν	\nu	ξ	\xi
	o	o	π	\pi	ϖ	\varpi	ρ	\rho
	ι	\iota	κ	\kappa	ϱ	\varrho	σ	\sigma
	ς	\varsigma	τ	\tau	υ	\upsilon	ϕ	\phi
	φ	\varphi	χ	\chi	ψ	\psi	ω	\omega
大写希腊字符	Γ	\Gamma	Δ	\Delta	Θ	\Theta	Λ	\Lambda
	Ξ	\Xi	Π	\Pi	Σ	\Sigma	Υ	\Upsilon
	Φ	\Phi	Ψ	\Psi	Ω	\Omega		
常用数学符号	$\aleph$	\aleph	$\prime$	\prime	$\forall$	\forall	$\exists$	\exists
	$\wp$	\wp	$\Re$	\Re	$\Im$	\Im	∂	\partial
	∞	\infty	∇	\nabla	$\surd$	\surd	$\angle$	\angle
	$\neg$	\neg	$\int$	\int	$\clubsuit$	\clubsuit	$\diamondsuit$	\diamondsuit
	$\heartsuit$	\heartsuit	$\spadesuit$	\spadesuit				
二元运算符号	$\pm$	\pm	$\cdot$	\cdot	$\times$	\times	$\div$	\div
	$\circ$	\circ	$\bullet$	\bullet	$\cup$	\cup	$\cap$	\cap
	$\vee$	\vee	$\wedge$	\wedge	$\otimes$	\otimes	$\oplus$	\oplus
关系数学符号	$\leq$	\leq	$\geq$	\geq	$\equiv$	\equiv	$\sim$	\sim
	$\subset$	\subset	$\supset$	\supset	$\approx$	\approx	$\subseteq$	\subseteq
	$\supseteq$	\supseteq	$\in$	\in	$\ni$	\ni	$\propto$	\propto
	$\mid$	\mid	$\perp$	\perp				
箭头符号	$\leftarrow$	\leftarrow	$\uparrow$	\uparrow	$\Leftarrow$	\Leftarrow	$\Uparrow$	\Uparrow
	$\rightarrow$	\rightarrow	$\downarrow$	\downarrow	$\Rightarrow$	\Rightarrow	$\Downarrow$	\Downarrow
	$\leftrightarrow$	\leftrightarrow	$\updownarrow$	\updownarrow				

```
>> t=['\partial(f_ip)/\partialt=-\Sigma_{i=1}^n\partial(f_ip)/', ...
     '\partialx_i + 0.5\Sigma_{i=1}^n\Sigma_{j=1}^n',...
     '\partial^2(b_{ij}p)/\partialx_i\partialx_j'];
   gtext(t);
   gtext('Y(\omega)=\int_0^\infty y(t)e^{-j\omegat}dt');
```

从上面的例子可以看出，这样需要两次用鼠标选择坐标。事实上，上面的命令可以由下面更简单的语句取代：

```
>> t=['\partial(f_ip)/\partialt=-\Sigma_{i=1}^n\partial(f_ip)/',...
      '\partialx_i + 0.5\Sigma_{i=1}^n\Sigma_{j=1}^n',...
      '\partial^2(b_{ij}p)/\partialx_i\partialx_j'];
   tt=str2mat(t,'Y(\omega)=\int_0^\infty y(t)e^{-j\omegat}dt');
   [x,y]=ginput(1); text(x,y,tt);
```

和标准的 TeX 不同，这里由 \ 引导的数学符号命令和其他字符串间可以不加空格，以免在最终显示上出现不必要的间隔。最终的显示结果将如图 2-12 所示。

$\partial(f_ip)/\partial t=-\Sigma_{i=1}^n\partial(f_ip)/\partial x_i + 0.5\Sigma_{i=1}^n\Sigma_{j=1}^n\partial^2(b_{ij}p)/\partial x_i\partial x_j$

$Y(\omega)=\int_0^\infty y(t)e^{-j\omega t}dt$

图 2-12 类似 TeX 的显示格式

由于直接采用类似 TeX 格式显示字符串的局限性，由 Ben Hinkle 开发的 teximage.m 函数能更好地将 LaTeX 支持的命令在图形窗口上叠印出来，其原理是：利用 MikTeX 程序将字符串的数学格式变换成图形文件，叠印到 MATLAB 的图形上。哈工大土木工程学院的王刚博士对该程序进行了适当修改，使得能结合 CJK 处理中文，这里我们将修改后的文件改名为 teximage_c.m。

下面的语句可以得出如图 2-13 所示的图形显示。

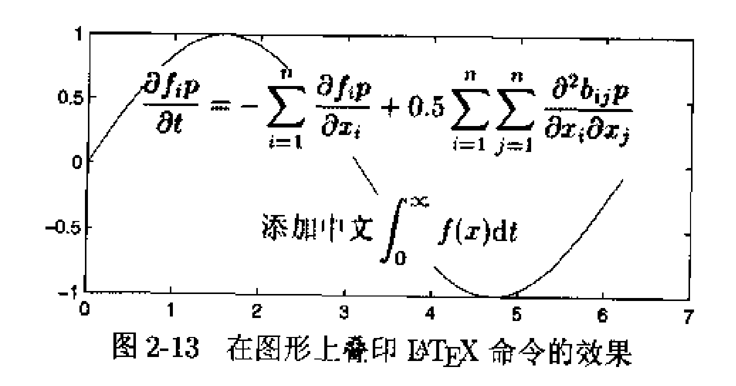


图 2-13 在图形上叠印 LaTeX 命令的效果

```
>> t=0:.1:2*pi; y=sin(t); plot(t,y) % 绘制正弦曲线
   txt=['\displaystyle \frac{\partial f_i p}{\partial t}=',...
     '-\sum_{i=1}^n\frac{\partial f_i p}{\partial x_i} + ',...
     '0.5\sum_{i=1}^n\sum_{j=1}^n',...
     '\frac{\partial^2 b_{ij} p}{\partial x_i\partial x_j}'];
   txt1='\displaystyle \mbox{添加中文} \int_0^\infty f(x)\mbox{d} t';
   teximage_c(txt); teximage_c(txt1);
```

遗憾的是，该图形生成的 eps 文件较大，且是基于位图的，效果不甚理想，所以，在 LaTeX 排版中，可以嵌入不使用 teximage_c.m 的图形，再用 overpic 包叠印真正的 LaTeX 字符串。

2.6.5 MATLAB 的图形可视编辑工具

MATLAB 从其 5.3 版本开始在图形窗口中提供了新的可视图形编辑工具，而在早期的版本中没有这样的可视功能，很多用户使用作者编写的图形可视处理工具 Graf_Tool[53] 来完成图形编辑的工作。

MATLAB 6.1 版本中的图形窗口如图 2-14 所示。可以看出，在这个窗口上提供了一个工具栏，该程序允许用户在图上添加标记字符、直线和箭头等。其实，图形窗口中提供了强大的图形编辑功能。例如打开 File 菜单，其内容如图 2-15(a) 所示。选中其中的 File | Property Editor 菜单则将打开属性编辑程序界面，如图 2-15(b) 所示。双击该界面上部的 figure (#1) 标志，则可以得出如图 2-16(a) 所示的显示，其中列出了当前图形窗口下的全部子对象，其中有一项为 axes，即坐标轴。再双击坐标轴标志，则得出如图 2-16(b) 所示的下一级子对象。选中其中一个具体的对象属性，就可以直接用本属性编辑程序处理了。可见，编辑对象的属性是很简单的事，很多属性的设置都可以通过可视的方式完成。

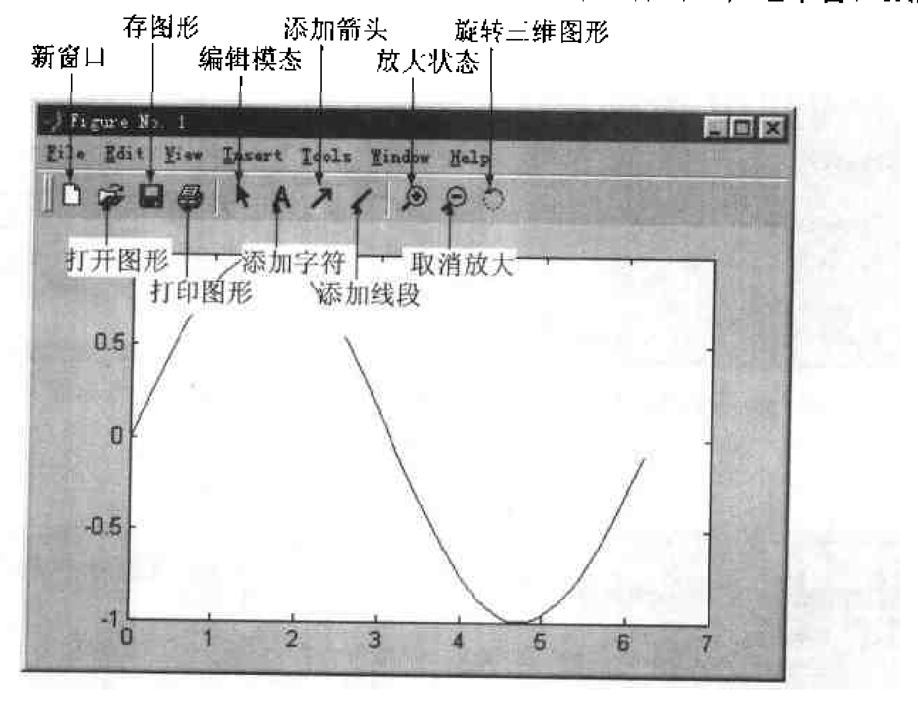
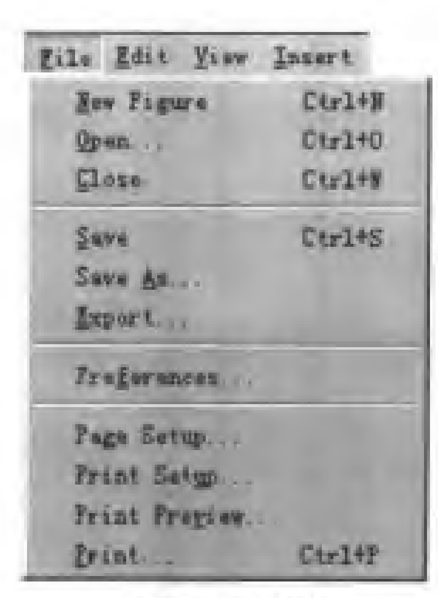


图 2-14 MATLAB 6.1 版本的图形窗口

单击图形窗口工具栏中的编辑图标 (箭头) 则可以进入编辑状态，用户可以选取图形窗口中的任意一个对象，这时该对象上会出现选中标志。我们可以容易地移动选中的字符对象。还可以选择添加文字工具、划线工具和箭头绘制工具对图形进行编辑。一个演示性的编辑结果如图 2-17 所示。

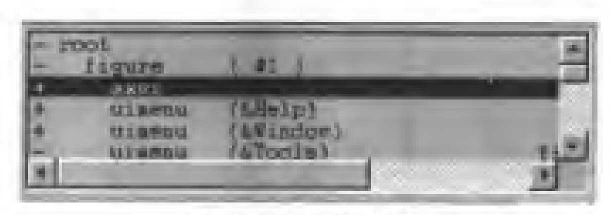
图形编辑程序的 Tools 菜单的功能也是很强大的，该菜单的内容如图 2-18(a) 所示。若选择了图形窗口中的一条曲线或直线，则再选择 Tools | Line Properties 菜单项，则会打开曲线对象属性设置对话框，如图 2-18(b) 所示。若选择了一个字符标记，再选择 Tools | Text Properties 菜单项，则将打开一个标准的字体设置对话框。用户可以在该对话框下设

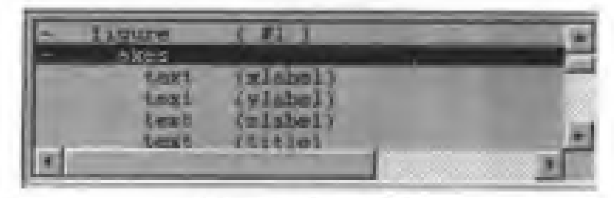


(a) File 菜单内容

(b) 图形属性编辑界面

图 2-15 File 菜单及图形属性编辑界面





(a) 图形窗口的子对象

(b) 坐标系的子对象

图 2-16 属性编辑窗口中的对象关系

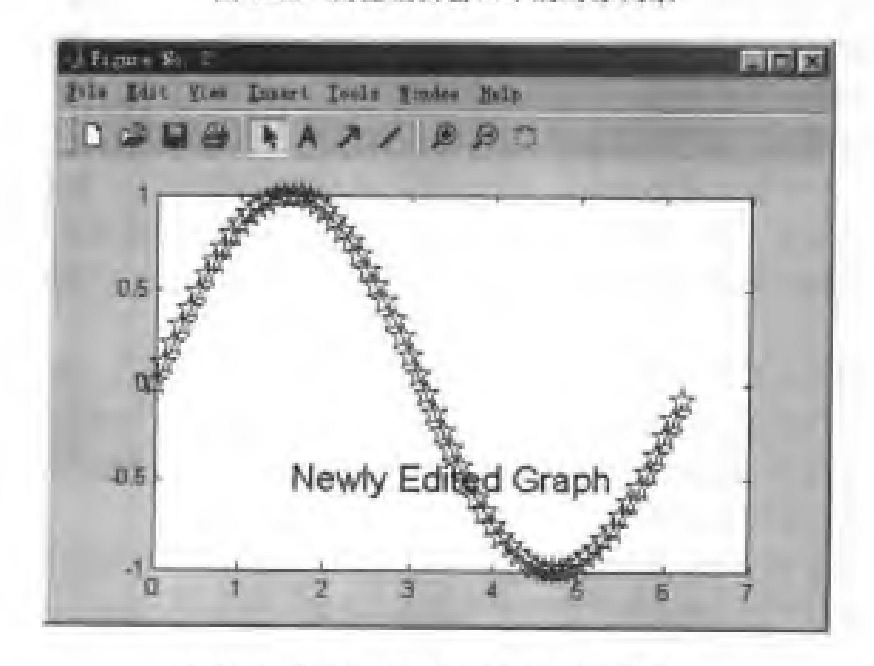
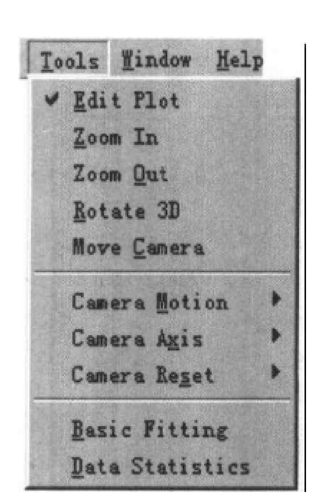
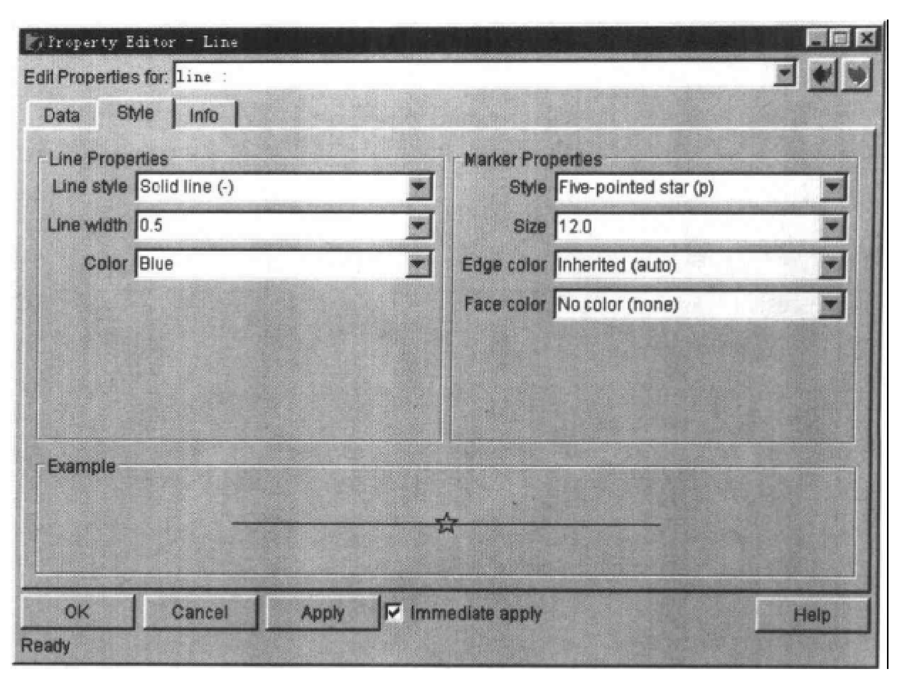


图 2-17 若干对象属性修改后的图形窗口




(a) Tools 菜单



(b) 线对象属性设置对话框

图 2-18 Tools 菜单及曲线属性设置对话框

置选定字符的字体。

选择 Tools | Axes Properties 菜单项，则将打开坐标轴属性编辑对话框，如图 2-19 所示。可以看出，这里包含了坐标轴属性的各种各样信息，用户可以方便地按照自己的愿望设置坐标轴的属性。

菜单项 Tools | Add | Axes 允许用户用鼠标在当前窗口上建立一个新的坐标系，这使得 MATLAB 图形处理与应用变得更加方便与可靠。

图形编辑程序允许用户按照它提供的格式将图形窗口中的图形保存起来，以备以后使用。它对应的文件后缀名为 **fig**。当然还可以用工具栏中的打开文件按钮读入 **fig** 文件。此外该程序还提供了三维图形的视角变换功能，在后面介绍三维图形时再叙述三维图形的旋转方法。

2.6.6 特殊图形绘制函数及举例

除了标准的二维曲线绘制之外，MATLAB 还提供了具有各种特殊意义的图形绘制函数，其常用调用格式如表 2-3 所示，其中参数 x，y 分别表示横纵坐标绘图数据，c 表示颜色选项，u，l 表示误差图的上下限向量。当然随着输入参数个数及类型的不同，各个函数的绘图形式也有所区别。下面将通过一个例子来演示各个绘图函数的效果。

【例 2.20】 可以用 subplot() 函数将整个图形窗口分割成 2×2 个子图形部分，然后在每个部分用不同的语句绘制出不同曲线。如果给出下面的命令，则可以得出如图 2-20 所示的曲线。

```
>> t=-pi:0.3:pi; y=1./(1+exp(-t));
```

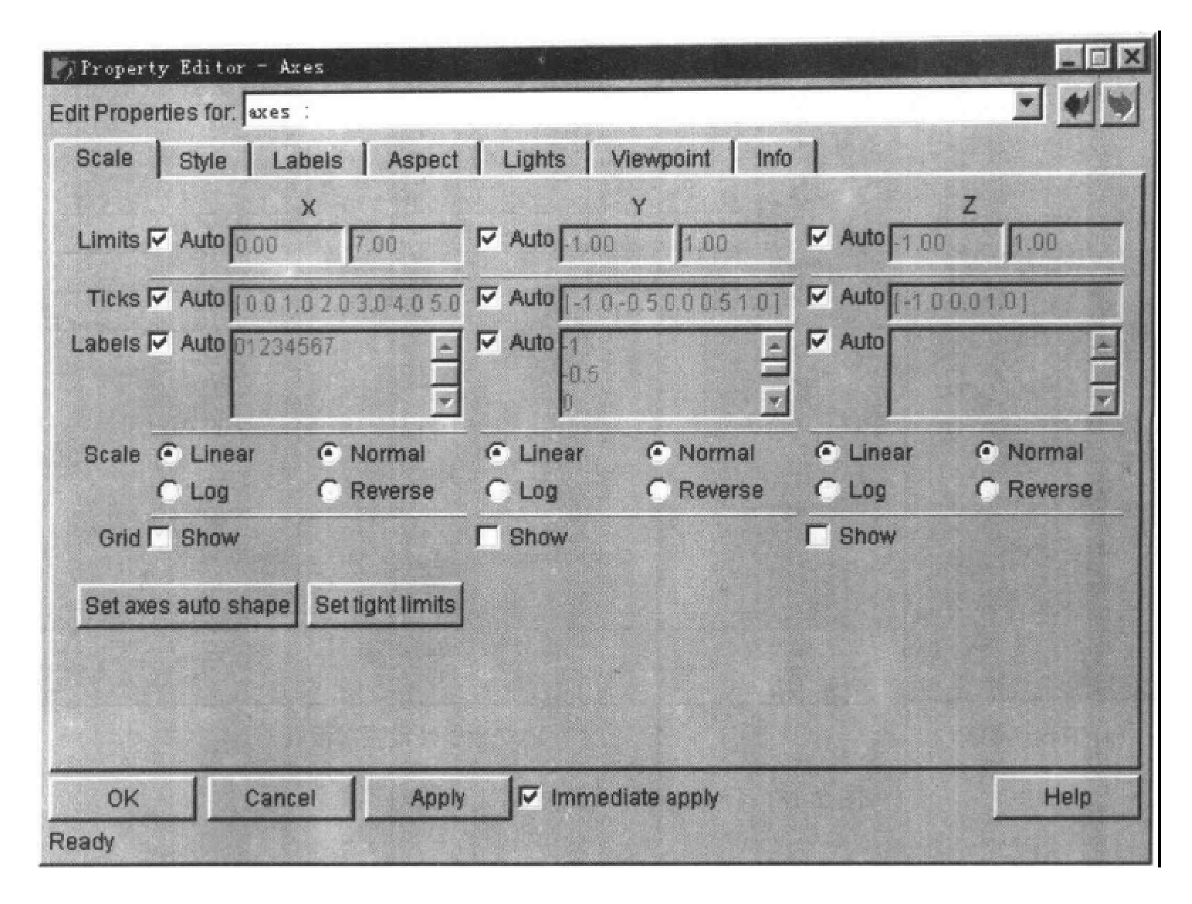


图 2-19 坐标轴属性编辑对话框

表 2-3 MATLAB 提供的特殊二维曲线绘制函数

函数名	意 义	常 用 调 用 格 式
bar()	二维条形图	bar(x, y)
comet()	彗星状轨迹图	comet(x,y)
compass()	罗盘图	compass(x, y)
errorbar()	误差限图形	errorbar(x, y, l, u)
feather()	羽毛状图	feather(x, y)
fill()	二维填充函数	fill(x, y, c)
hist()	直方图	hist(y, n)
loglog()	对数图	loglog(x, y)
polar()	极坐标图	polar(x, y)
quiver()	磁力线图	quiver(x, y)
stairs()	阶梯图形	stairs(x, y)
stem()	火柴杆图	stem(x, y)
semilogx()	半对数图	semilogx(x, y),semilogy(x, y)

```
subplot(221), plot(t,y); title('plot(t,y)')
subplot(222), stem(t,y); title('stem(t,y)')
subplot(223), semilogy(t,y); title('semilogy(t,y)')
subplot(224), stairs(t,y); title('stairs(t,y)')
```

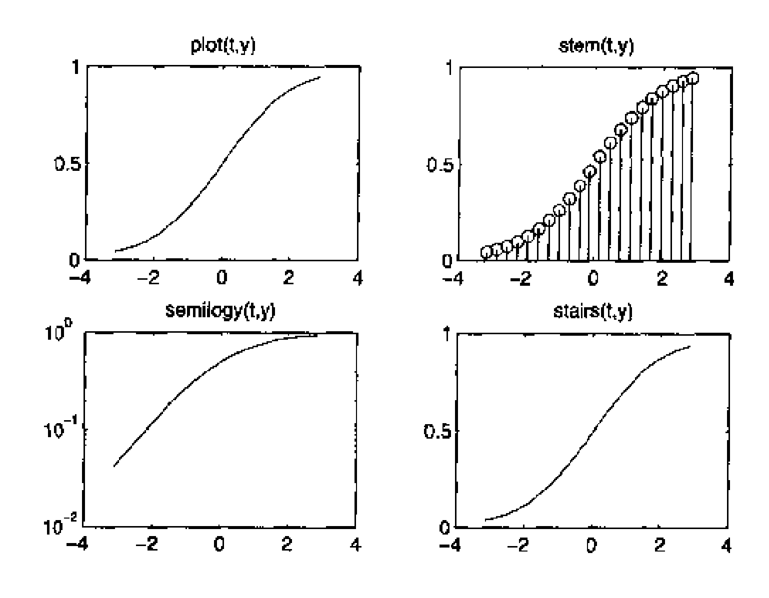


图 2-20 三维空间的曲线绘制

2.6.7 给定函数的曲线绘制

如果给定了函数的显式表达式，可以先设置自变量向量，然后根据表达式计算出函数向量，从而用 plot() 等函数绘制出来，如果函数由隐函数形式给出，例如 $x^2+3y^2=5$，则很难用上述方式绘制出图形。MATLAB 中提供了一个 ezplot() 绘制隐函数，这里将通过几个例子来演示该函数的调用。

【例 2.21】 前面例子中的隐函数 $x^2+3y^2=5$ 可以由如下命令直接绘制出来，就可以直接绘制出如图 2-21 (a) 所示的椭圆。

```
>> ezplot('x^2+3*y^2-5')  % 绘制隐函数图形
   axis([-4,4,-4,4])       % 设置坐标轴范围 (默认的不甚理想)
```

如果只想绘制出 $x\in(-\pi/4,\pi)$, $y\in(-1,3)$ 之间的部分，则可以由下面的命令得出如图 2-21 (b) 所示的结果。

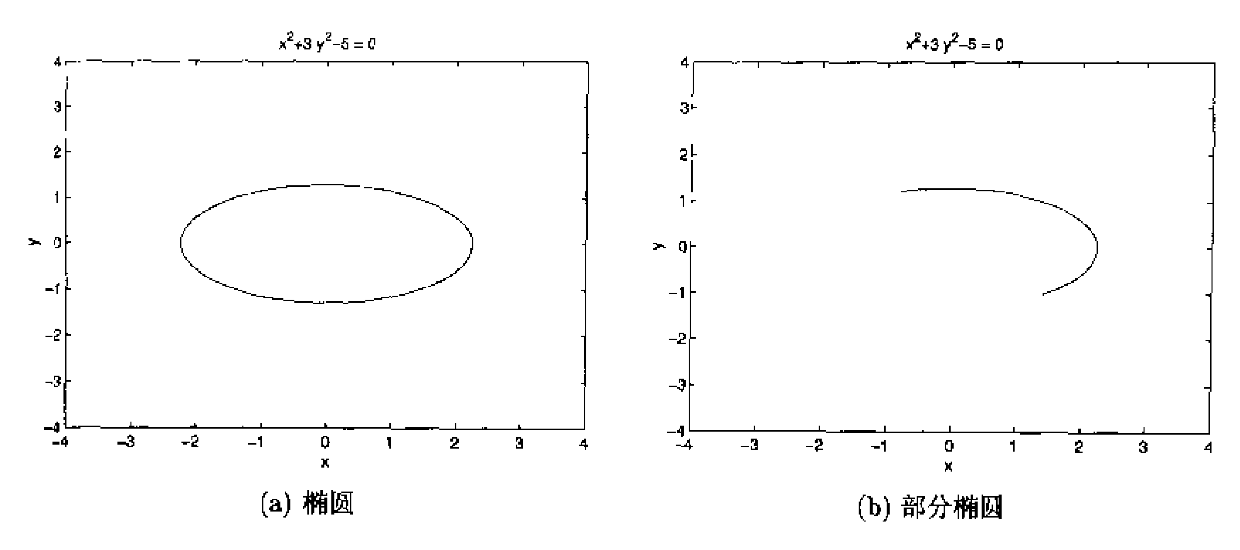
```
>> ezplot('x^2+3*y^2-5',[-pi/4,pi,-1,3])  % 绘制隐函数图形
   axis([-4,4,-4,4])       % 设置坐标轴范围 (默认的不甚理想)
```

考虑下面更复杂的函数

$$\frac{1}{y}-\ln(y)+\ln(-1+y)+x-\sin(x)=0$$

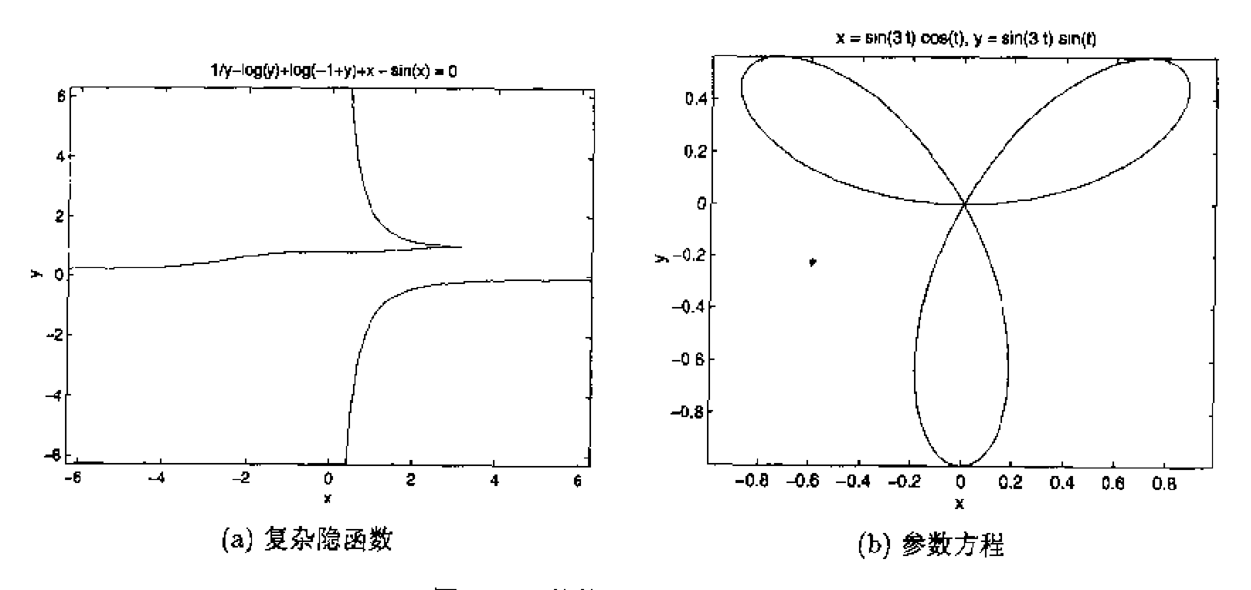
可以用 ezplot() 函数直接绘制其曲线，如图 2-22 (a) 所示。

```
>> ezplot('1/y-log(y)+log(-1+y)+x - sin(x)')
```



(a) 椭圆　(b) 部分椭圆

图 2-21　由 ezplot() 函数直接绘制的椭圆



(a) 复杂隐函数　(b) 参数方程

图 2-22　其他形式函数的绘制

假设已知参数方程

$$x=\sin 3t\cos t,\ \ y=\sin 3t\sin t,\ \ t\in(0,\pi)$$

下面语句可以得出如图 2-22 (b) 所示的曲线。在该函数调用时还可以不给出 t 变量的范围。

```
>> ezplot('sin(3*t)*cos(t)','sin(3*t)*sin(t)',[0,pi])
```

2.7 三维图形的绘制方法

2.7.1 三维曲线的绘制方法

首先，和原来的二维图形相对应，MATLAB 提供了 plot3() 函数，它允许用户在

一个三维空间内绘制出三维的曲线，该函数的调用格式为：

```
plot3(x, y, z, 选项)
```

其中 x, y, z 分别为维数相同的向量，分别存储曲线的三个坐标的值；而这里所使用的选项和 plot() 函数是一致的，它可以定义曲线的线型、颜色等信息，具体的内容可以参见表 2-1。

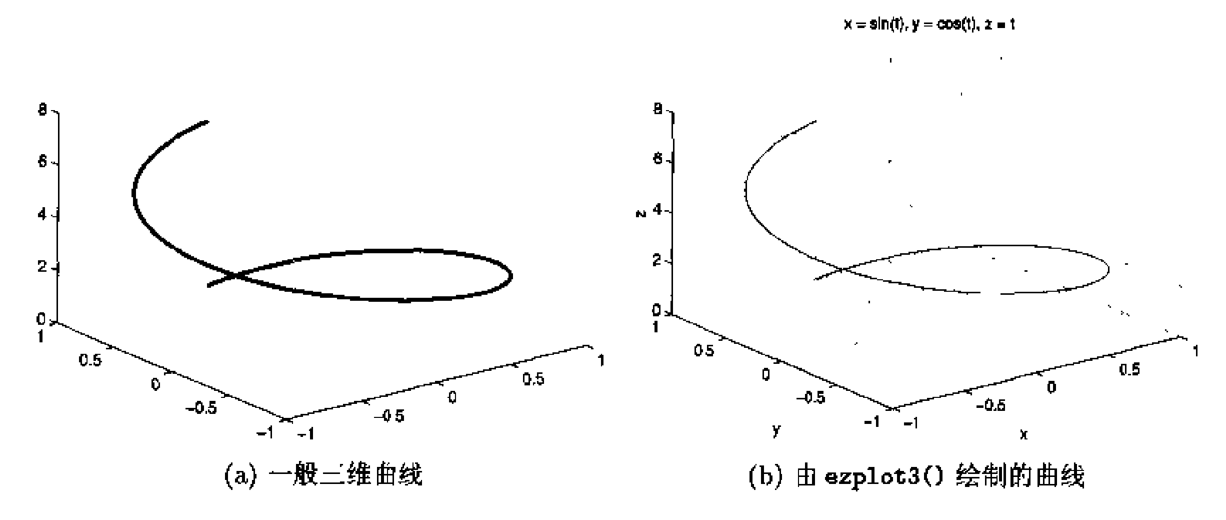
〖例 2.22〗 假设有一个时间向量 t，对该向量进行下列运算则可以构成三个坐标的值向量

$$x = \sin(t),\ \ y = \cos(t),\ \ z = t$$

则如果想用绿色的粗实线绘制此图形，就应该键入下面的程序段

```
>> t=0: pi/50: 2*pi;
   x=sin(t); y=cos(t); z=t; h=plot3(x, y, z, 'g-')
   set(h,'LineWidth',4*get(h,'LineWidth'))
```

由上面程序段绘制出来的三维曲线如图 2-23 (a) 所示。



(a) 一般三维曲线　　(b) 由 ezplot3() 绘制的曲线

图 2-23　三维空间的曲线绘制

该三维曲线还可以由下面的语句直接绘制出来，如图 2-23 (b) 所示，可见二者绘制的结果完全一致。

```
>> ezplot3('sin(t)','cos(t)','t',[0,2*pi])
```

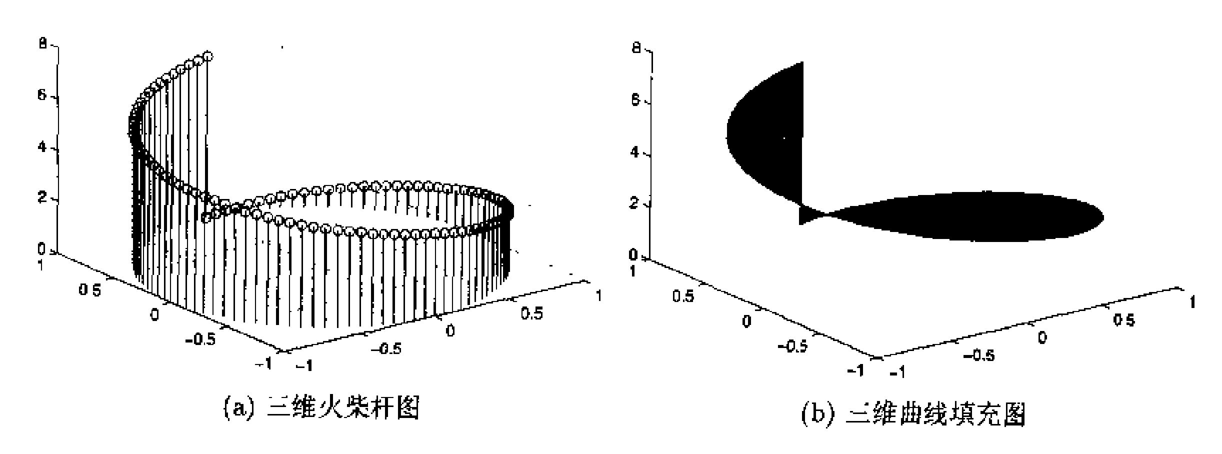
可以仿照二维 stem() 函数的形式绘制三维火柴杆图，并在坐标轴上加网格，得出的图形如图 2-24 (a) 所示。

```
>> grid on, stem3(x, y, z),
```

还可以使用下面的命令

```
>> grid off, fill3(x,y,z,'g'),
```

来绘制填充的三维曲线，并除去网格显示，如图 2-24 (b) 所示。



(a) 三维火柴杆图　　(b) 三维曲线填充图

图 2-24 不同三维曲线绘制函数

2.7.2 三维曲面的绘制方法

任意给定的数据均可以由三维曲面的方式显示，在MATLAB 下允许使用 mesh() 函数来绘制三维表面网格图。函数的调用格式可以写成：

```
mesh(x, y, z, c)
```

其中 x 和 y 分别为构成该曲面的 x 和 y 向量或矩阵；z 为高度矩阵；c 为颜色矩阵，表示在不同的高度下的颜色范围。如果省略 c 选项，则 MATLAB 会自动地假定 c=z，亦即颜色的设定是正比于图形的高度的，这样就可以得出层次分明的三维图形。

〖例 2.23〗 考虑下面给出的二元函数，在 x, y 平面内选择一个区域，然后绘制出

$$z = f(x, y) = (x^2 - 2x)e^{-x^2-y^2-xy}$$

的三维表面图形。

首先可以调用 meshgrid() 函数生成 x 和 y 平面的网格表示。该函数的调用意义十分明显，即可以产生一个横坐标起始于 −3，中止于 3，步距为 0.1，纵坐标起始于 −2，中止于 2，步距为 0.1 的网格分割。然后由上面的公式计算出曲面的 z 矩阵。最后调用 mesh() 函数来绘制曲面的三维表面网格图形。

```
>> [x,y] = meshgrid(-3:0.1:3,-2:0.1:2);
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y); mesh(x,y,z)
```

事实上，由 meshgrid() 函数生成的 x 和 y 均是矩阵。可以看出，在计算 z 矩阵时大量地采用了点运算，通过给定的函数计算出高度矩阵 z。当然这样的语句自动产生的坐标不一定很理想，所以可以调用 axis() 函数来重新设定坐标系范围：

```
>> axis([-3,3,-2,2,-0.7,1.5])
```

最后得出的曲线如图 2-25(a) 所示。

由图 2-25(a) 可以看出，在这种默认的状态下隐含的部分都没有绘制出来，这是合乎正常视觉的绘制方法。如果用户实在想绘制出隐含的部分，则可以调用 hidden off 命令来进行处理，

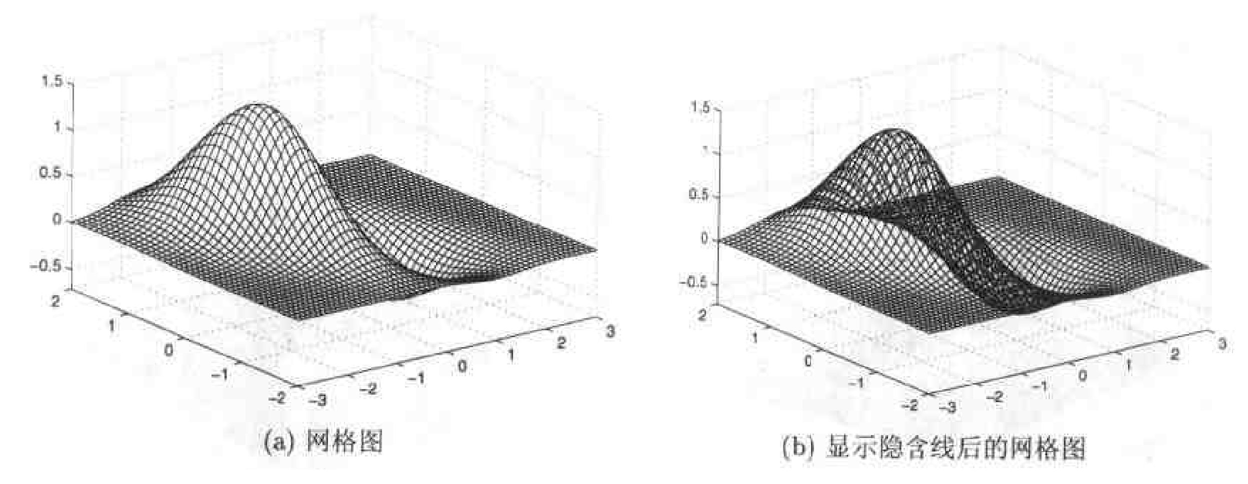


图 2-25 由 mesh() 函数得出的三维网格图形

这时得出的图形如图 2-25(b) 所示，从图中可以看出，所有的隐含网格也都同时绘制出来了。如果用户还想恢复到默认的状态，则可以调用 hidden on 命令来进行设置。

如果将前面用到的 mesh() 函数用 surf(x, y, z) 函数取代，则绘制出来的表面图形如图 2-26(a) 所示。MATLAB 中提供了一个 colorbar 命令，可以在显示的三维图旁边显示出指示高度的彩色条，使得三维表面图更具可读性。本例中的表面图使用了该命令之后，则得出如图 2-26 (b) 所示的显示。

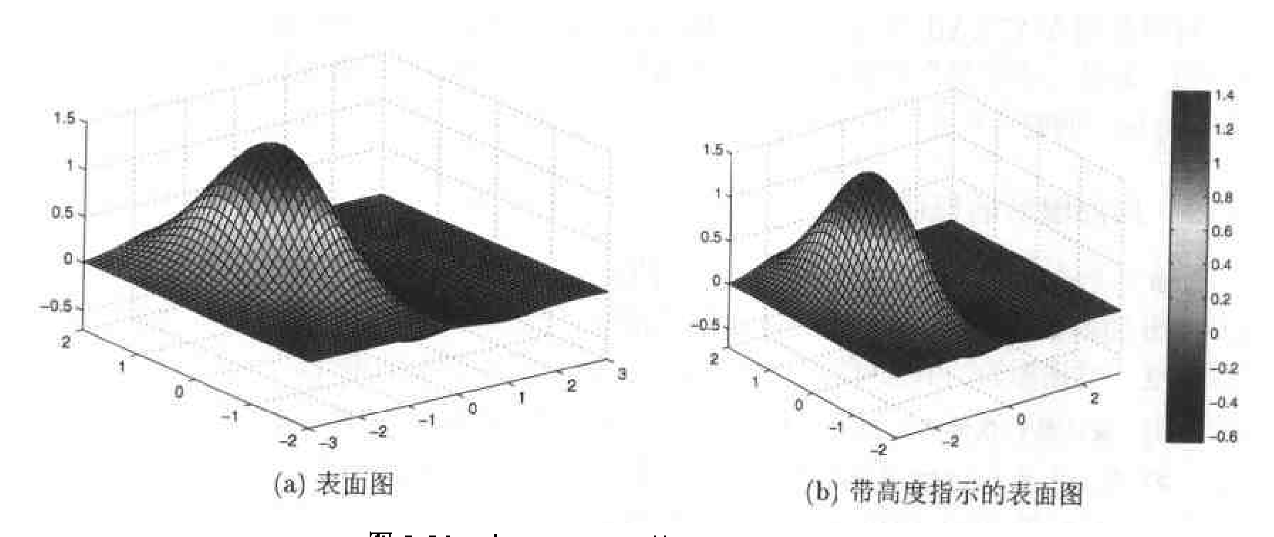


图 2-26 由 surf() 函数得出的三维表面图

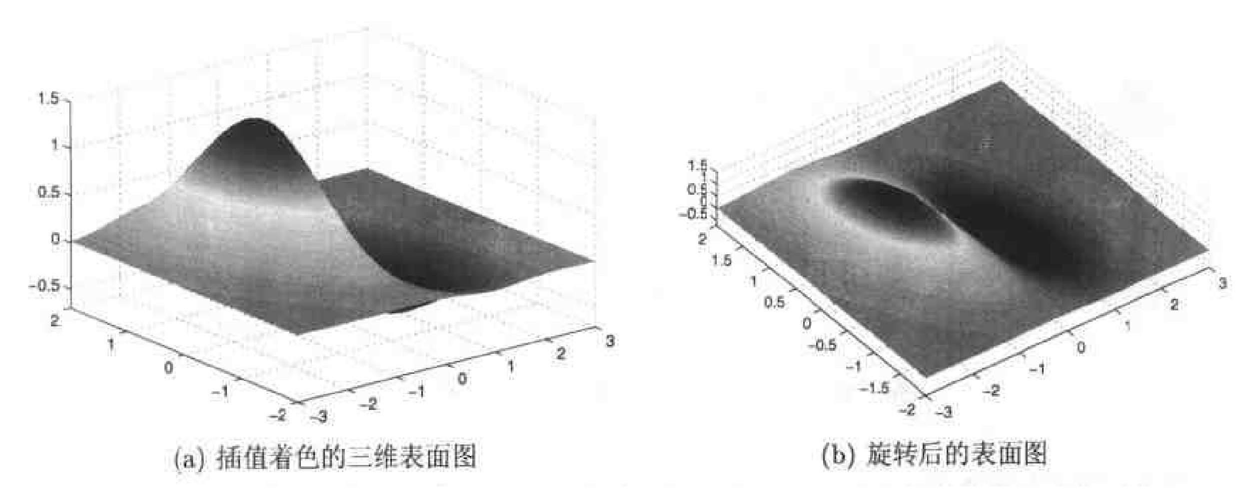
使用 MATLAB 中提供的 ezmesh() 和 ezsurf() 函数也能得出同样的图形。

```
>> ezmesh('(x^2-2*x)*exp(-x^2-y^2-x*y)')
   figure; ezsurf('(x^2-2*x)*exp(-x^2-y^2-x*y)')
```

注意，在这里描述表达式时不再采用点运算的符号了。另外，这里的表达式应该是 z 的显式表达式，亦即 $z=f(x,y)$，否则不能绘制出三维曲面。

MATLAB 提供的 shading 命令可以设置表面图的各种着色方案，例如，该命令加

interp 选项，即使用 shading interp 命令，则将使得表面用插值的方式进行着色，使得图像表面用光滑的形式显示，如图 2-27 (a) 所示。



(a) 插值着色的三维表面图　　(b) 旋转后的表面图

图 2-27　三维表面图的进一步处理

除了该选项外，还可以使用 flat 选项 (其表面上每个网格中用相同的颜色着色，并将网格线隐含起来)、faceted 选项 (采用默认的方式着色) 等。

可以使用 MATLAB 工具栏中提供的图形旋转按钮来任意旋转三维图形。单击工具栏中的“旋转三维图形”按钮，就可以自如地旋转得出的三维图形。例如可以得出如图 2-27 (b) 所示的旋转结果。

2.7.3　局部图形的剪切处理

NaN 常数在图形处理中是一个很有意思的数值。如果不想要图形中的某个部分，只需把该部分的函数值设置成 NaN 即可。这样在绘制三维图形时，值为 NaN 的部分将不显示出来。在二维图形绘制中也可以采用这样的技术剪切掉不需要的部分。

〖例 2.24〗 假设想剪裁掉例 2.23 的三维图中 $x \leq 0$ 和 $y \leq 0$ 的部分，则可以使用如下命令：

```
>> [x,y] = meshgrid(-3:0.1:3,-2:0.1:2);
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);
   ii=(x<=0)&(y<=0); z1=z; z1(ii)=NaN;
   surf(x,y,z1), shading flat; axis([-3 3 -2 2 -0.7 1.5])
```

由此得出效果图如图 2-28 所示。

2.7.4　图像显示与处理

在 MATLAB 中提供了图像文件读取函数 imread()，该函数可以将图像文件按三维数组的形式读入 MATLAB 的工作空间，这样的三维数组可以用 image() 函数绘制出来。

〖例 2.25〗 假设有一个图像文件 tiantan.jpg，则可以用

```
>> W=imread('tiantan.jpg');
```

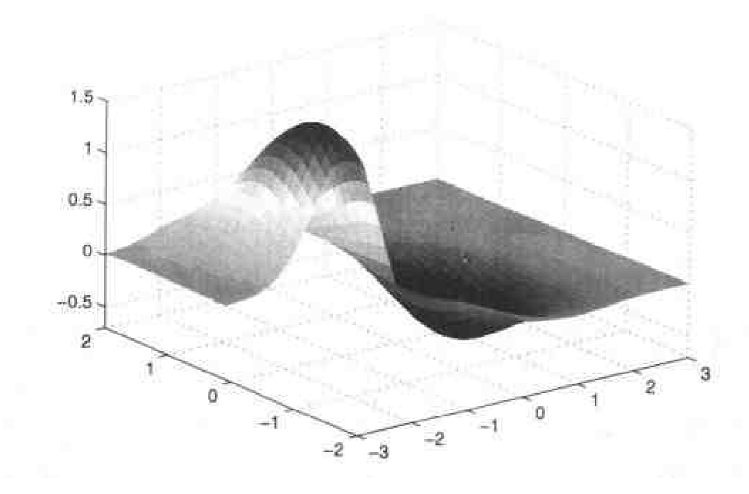


图 2-28 三维表面图的剪切与效果

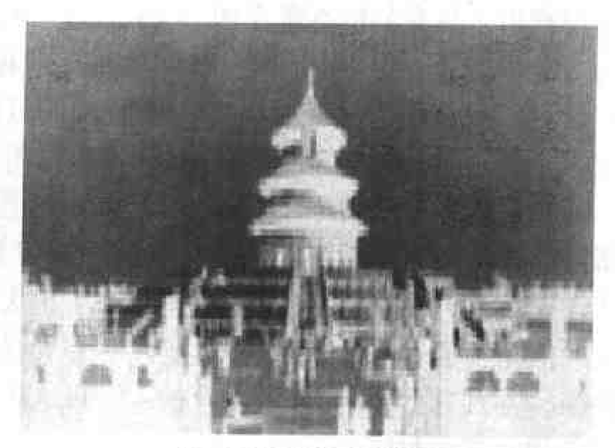
语句将该图像文件读入 MATLAB 工作空间，这样该图像就能用工作空间中的三维数组 W 表示，该三维数组实际上由三层的矩阵叠放而成，这三层分别存放红、绿、蓝三原色的分量，该三维数组实际上是一个无符号 8 位整型数的数组，每个元素的取值范围为 0~255。在当前的 MATLAB 版本中，能根据图像文件的后缀名判定图像类型，从而自动读入该图像。

图像读入后，可以用下面的 MATLAB 语句将该图像在图形窗口上绘制出来，如图 2-29 (a) 所示，

```
>> image(W), set(gca,'Visible','off')
```



(a) 图像绘制效果



(b) 反色后的图像效果

图 2-29 图像绘制与简单处理

其中 set(gca,'Visible','off') 命令将坐标轴上的标度隐含起来，使得它只显示图像本身。下面考虑如何将该图像反色，得出“照相底片”的效果。从原理上来讲还是很简单的，用无符号 8 位整型数的最大值 255 剪去 W 数组的所有元素即可。所以可以尝试给出如下的命令

```
>> image(255-W)
??? Error using ==> -
```

```
Function '-' not defined for variables of class 'uint8'.
```

不幸的是，该命令得出错误结果，这是由于当前的 uint8 型数据不支持减法运算符，所以可以先将之变换成双精度数据，进行减法运算，再变换回 uint8 型数据即可。这样应该给出下面的命令，最终得出如图 2-29 (b) 所示的图像处理效果。

```
>> image(uint8(255-double(W))), set(gca,'Visible','off')
```

2.8 MATLAB 图形用户界面设计技术

对一个成功的软件来说，其内容和基本功能当然应是第一位的。但除此之外，图形界面的优劣往往也决定着该软件的档次，因为用户界面会对软件本身起到包装作用，而这又像产品的包装一样，所以能掌握 MATLAB 的图形界面设计技术对设计出良好的通用软件来说是十分重要的。

早期的 MATLAB 版本只提供了一个命令屏幕和一个图形屏幕，用户只能在这两个屏幕之间进行切换。如果用户在命令屏幕上给出一条 MATLAB 绘图命令，则要想得出相应的图形，MATLAB 会自动地切换到图形屏幕上，如果再给出一条命令，则会自动地切换到原来的命令屏幕。虽然 MATLAB 可以在两个屏幕之间进行自动的切换，但使用起来还是极其不便的，尤其想同时显示出多幅图形时更是如此。因此，由于受当时技术局限性的影响，MATLAB 早期的版本并不适于开发用户友好的图形界面程序。

随着 Windows 技术的发展，MATLAB 的用户及 The MathWorks 公司的开发者们逐渐意识到在多个窗口界面下运行 MATLAB的必要性和可行性。1992 年 The MathWorks 公司推出了具有创造性意义的 MATLAB 4.0 版本，并于次年正式推出了 MATLAB 4.0 版的 PC 机版本，以适应日益流行的 Microsoft Windows 环境下使用。MATLAB 4.0 版本一出现，立即引起了使用者和程序开发人员的极大兴趣，因为它使在其他语言环境下看起来十分复杂的 Windows 图形界面设计显得非常的容易和方便。

MATLAB 5.0 版的出现使 MATLAB 图形界面设计技术进入了一个新的阶段。该版本提供了一个实用的用户图形界面开发程序 Guide，然而在该版本中其功能很不完善，6.0 版中提供的 Guide 程序功能有了很大的改观，但有些地方也不甚理想，MATLAB 6.1 中增强了 Guide 程序的功能，它完全支持可视化编程，其方便程度类似于 Visual Basic。将它提供的方法和用户的 MATLAB 编程经验结合起来，可以很容易地写出高水平的用户界面程序。

在本节中先介绍 Guide 的使用方法，然后将举例说明用 MATLAB 语言如何容易地实现图形用户界面的设计，并介绍界面设计的应用。

2.8.1 图形界面设计工具 Guide

在 MATLAB 命令窗口中键入 guide 命令，则将得出如图 2-30 所示的设计窗口，其中右侧的窗口区域就是要设计窗口的雏形 (prototype)。

双击该窗口雏形，则将得出如图 2-31 (a) 所示的对话框，允许用户修改其中的内容

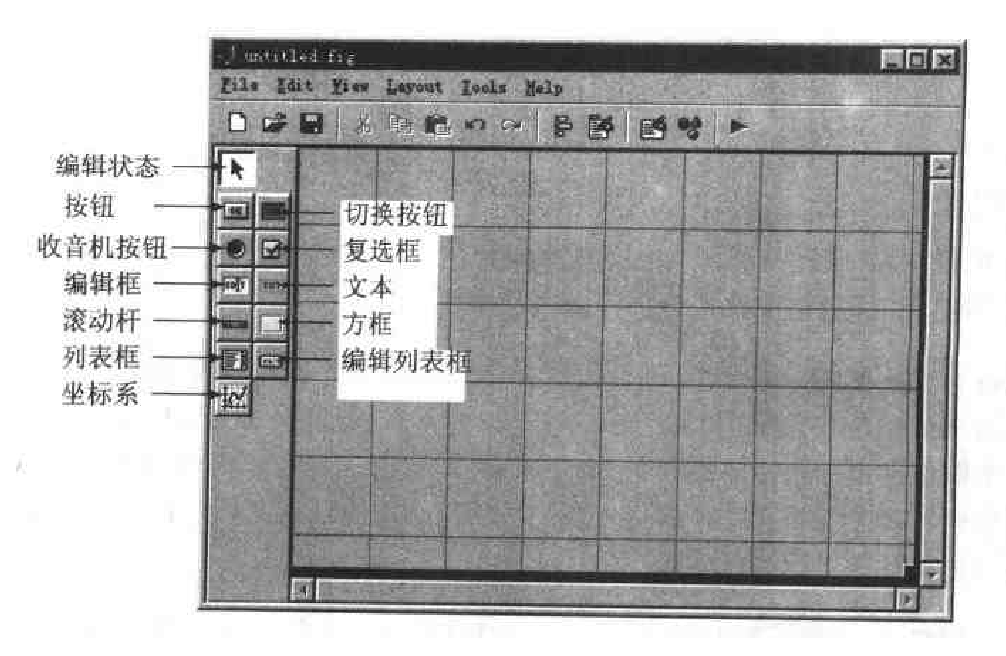
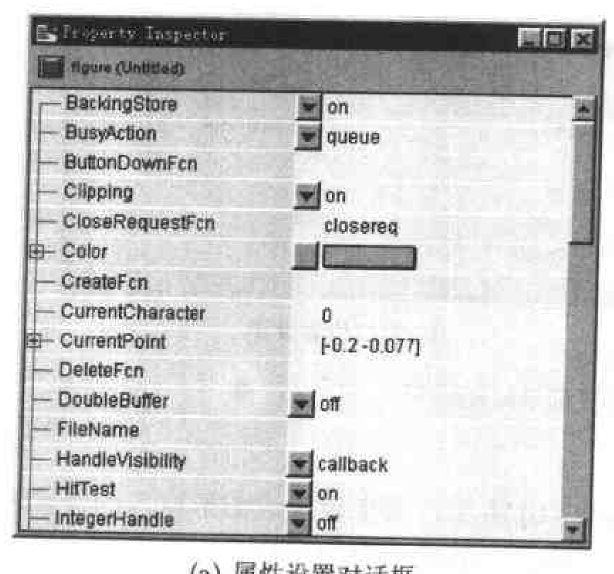
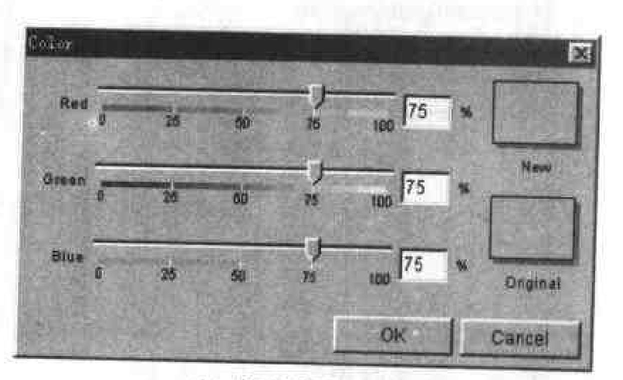


图 2-30 Guide 程序界面及说明

来改变该窗口的属性。例如，若想修改窗口的颜色，则可以在其中的 Color 栏目下单击其右侧的方框，这样将得出如图 2-31 (b) 所示的对话框，调整其红、绿、蓝颜色分量的配比，得出所需要的颜色，颜色修改完成后将立即在雏形窗口中显示出来。



(a) 属性设置对话框



(b) 颜色设置对话框

图 2-31 界面属性修改

可以看出，在该对话框中列出了有关窗口的众多属性，通常没有必要改变所有的属性，下面仅列出常用的属性：

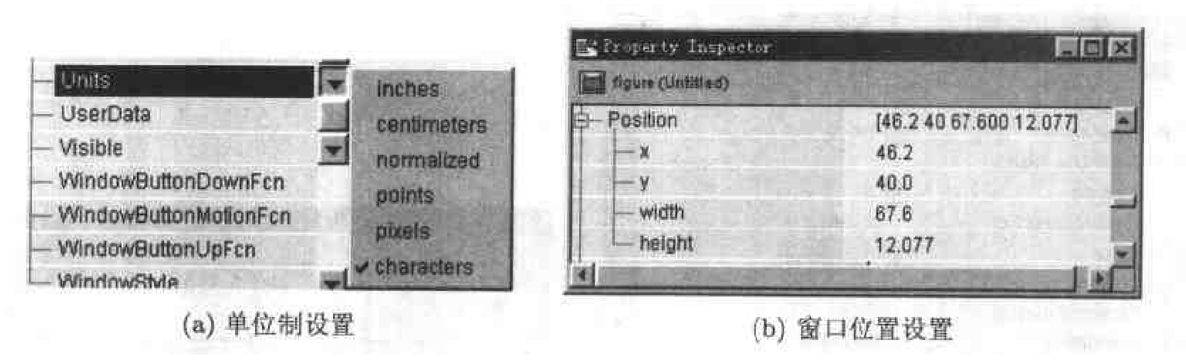
- MenuBar 属性 - 设置图形窗口菜单条形式，可选择 'figure' (图形窗口标准菜单) 或 'none' (不加菜单条) 选项。用户如果选中了 'none' 属性值，则在当前处理的窗

口内将没有菜单条，这时用户可以根据后面将介绍的 uimenu() 函数来加入自己的菜单条。如果用户选择了其中的 'figure' 选项值，则该窗口将保持图形窗口默认的菜单项 (其中相应的选项后面将进行介绍)。选择了 'figure' 选项后，还可以用 uimenu() 函数改变原来的标准菜单，或者添加新的菜单项。

- Name 属性 —— 设置图形窗口的标题栏中标题内容，它的属性值应该是一个字符串，在图形窗口的标题栏中将把该字符串内容填写上去。

- NumberTitle 属性 — 决定是否设置图形窗口标题栏的图形标号，它相应的属性值可选为 'on' (加图形标号) 或 'off' (不加标号)，若选择了 'on' 选项则会自动地给每一个图形窗口标题栏内加一个 Figure No *: 字样的编号，即使该图形窗口有自己的标题也同样要在前面冠一个编号，这是 MATLAB 的默认选项，若选择 'off' 选项则不再给窗口标题进行编号显示了。

- Units 属性 —— 除了默认的像素点单位 'pixels' 之外，还允许用户使用一些其他的单位，如 'inches' (英寸)、'centimeters' (厘米)、'normalized' (归一值，即 0 和 1 之间的小数) 等，这种设定将影响到一切定义大小的属性项 (如后面将介绍的 Position 属性)。Units 属性也可以通过属性编辑程序界面来设定，例如选择 Units 属性时，在属性值处将出现一个列表框，如图 2-32 (a) 所示，用户可以从中选择希望的属性值。



(a) 单位制设置 (b) 窗口位置设置

图 2-32 窗口的位置和单位制设置

- Position 属性 —— 该属性的内容如图 2-32 (b) 所示，用来设定该图形窗口的位置和大小。其属性值是由 4 个元素构成的 1×4 向量，其中前面两个值分别为窗口左下角的横纵坐标值，后面两个值分别为窗口的宽度和高度，其单位由 Units 属性设定。设置 Position 属性值的最好方法是：首先关闭属性设置对话框，直接对该窗口进行放大或缩小，然后再打开属性编辑程序。这样在 Position 一栏中就自动地填写上用户设置的值了。

- Resize 属性 —— 用来确定是否可以改变图形窗口的大小。它有两个参数可以使用：'on' (可以调整) 和 'off' (不能调整)，其中的 'on' 选项为默认的选项。

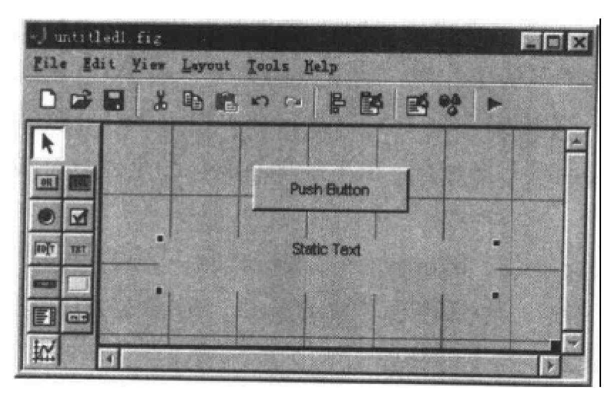
- **Toolbar 属性** — 表示是否给图形窗口添加可视编辑工具条，其选项为 'none' (无工具条)、'figure' (标准图形窗口编辑工具条) 和 'auto' (自动)。一般若想对图形进行可视修改，最好将此选项设置为 'figure'。
- **Visible 属性** — 它用来决定建立起来的窗口是否处于可见的状态，对应的属性值为 'on' 和 'off' 两种，其中 'on' 为 MATLAB 的默认属性值。
- **Pointer 属性** — 用来设置在该窗口下指示鼠标位置的光标的显示形式，用户还可以用 PointerShapeCData 属性自定义光标的形状。

注意，其中某些属性不能在编辑状态下正确显示，例如 Name 和 NumberTitle 属性等，但在程序运行时应该没有问题。

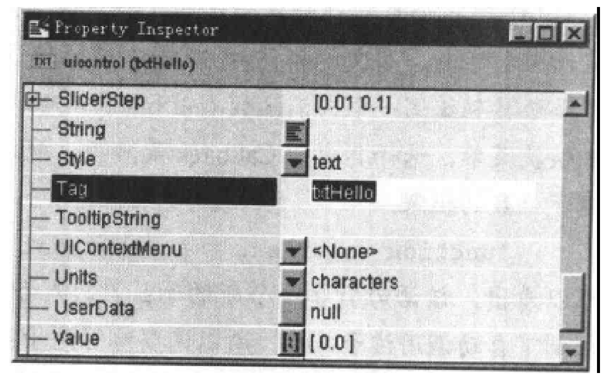
在该界面的左侧工具栏中，提供了各种各样的控件，可以通过左击的方式选中其中一个控件，这样就可以在右侧的雏形窗口中绘制出这个控件。可以通过这样的方法在雏形窗口上绘制出各种控件，实现所需图形用户界面的设计。下面用简单例子来演示图形用户界面的设计方法。

〖例 2.26〗 考虑在一个窗口上添加一个“按钮”控件和一个用于字符显示的“文本”控件，并在按下该按钮时，在文本控件上显示“这是我的第一个界面”字样。具体的设计步骤如下:

(1) 在雏形窗口中绘制出这两个控件，如图 2-33 (a) 所示。



(a) 将所需控件绘制出来



(b) 修改控件属性

图 2-33 界面设计及修改

(2) 我们需要修改文本控件的属性，双击其图标，可以将其 String 属性设置为空字符串，表示在按下按钮前不显示任何信息。另外应该给该控件设置一个标签，即设置其 Tag 属性，以便在后面编程时能容易地找到其句柄。在这里可以将其设置为 txtHello，如图 2-33 (b) 所示。注意，在设置标签时应该将其设置为独一无二的，以使得程序能容易地找到它，而不是同时找到其他控件。

(3) 建立了控件之后，可以将其存成 *.fig 文件，如将其存为 c2fgui1.fig，这时还将自动生成一个 c2fgui1.m 文件，其主要部分内容如下，在此还给出必要的说明。

```
function varargout = c2fgui1(varargin) %引导语句
if nargin == 0  % 如果调用时不给出任何附加参数，则直接打开该窗口
```

```
        fig = openfig(mfilename,'reuse'); % 读 *.fig 文件，绘制出所需窗口
        set(fig,'Color',get(0,'defaultUicontrolBackgroundColor')); % 默认颜色
        handles = guihandles(fig); % 得出所有的句柄
        guidata(fig, handles);      % 将句柄存到窗口的句柄下
        if nargout > 0              % 如果调用时需要返回参数，则将该窗口句柄返回
           varargout{1} = fig;
        end
     elseif ischar(varargin{1}) % 调用时若给出附加参数，则用户需自己编写相应程序
        try
           if (nargout)
              [varargout{1:nargout}] = feval(varargin{:});
           else
              feval(varargin{:});
           end
        catch
           disp(lasterr);
        end
     end
```

(4) 编写响应按钮动作的程序。分析原来的要求，可以看出，实际上需要编写的响应函数是在按钮按下时，将文本控件的 String 属性值设置成所需的值，即“这是我的第一个界面”，这就需要给按钮编写一个回调函数 (callback)，可以右击按钮控件，这样将得出一个如图 2-34 (a) 所示的快捷菜单，选择其中的 Callback 菜单项，就能自动打开 c2fgui1.m 程序编辑界面，并生成一个回调子函数的框架：

```
function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
```

可以看出，该语句引导一个新的子函数，在程序执行后，如果单击其中的按钮，则在 MATLAB 机制下自动调用该子函数，所以需要编写该子函数来完成指定的任务。

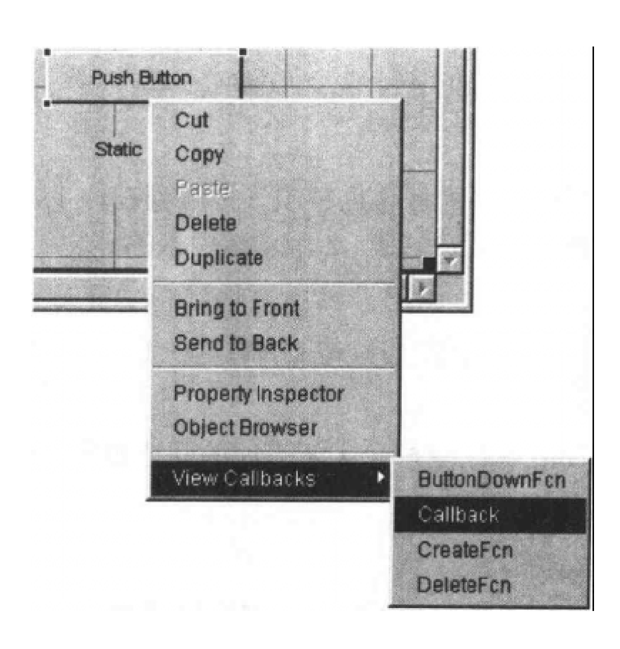
在该子函数中，允许返回可变数目的变量 (因为使用了 varargout 变量)，也允许使用可变数目的输入变量 (因为使用了 varargin 变量)，此外，h 为当前窗口的句柄，eventdata 是事件的代码，handles 是该窗口中的句柄集，其中的每个句柄可以由设置的标签直接访问。所以要解决前面要求的问题，只需编写如下的子函数

```
function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
set(handles.txtHello,'String','这是我的第一个界面');
```

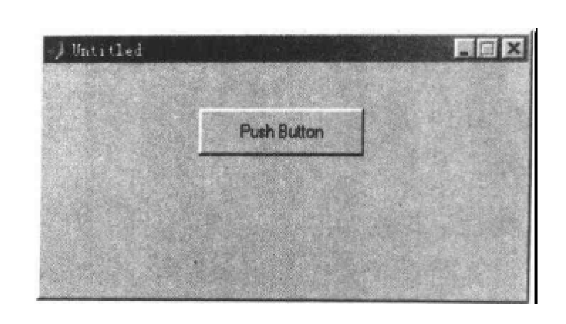
该子函数只有一个语句，即将句柄集的 handles.txtHello 分量 (该分量实际上就是文本控件的句柄) 'String' 的属性设置为所需的字符串，这样就完成了整个程序的设计。通过这个例子还可以看出设置标签的意义。

在 MATLAB 的命令窗口中键入 c2fgui1 就可以启动该函数，这将得出如图 2-34 (b) 所示的界面，按下其中的按钮，则界面将自动按期望发生变化，如图 2-34 (c) 所示。

MATLAB 图形用户界面设计的另一个值得注意问题是它所支持的各种回调函数，前




(a) 右键快捷菜单



(b) 程序启动界面



(c) 按钮按下的结果

图 2-34 界面设计简单演示

面已经演示过，所谓的回调函数就是，在对象的某一个事件发生时，MATLAB 内部机制允许的自动调用的函数，常用的回调函数包括：

- `CloseRequestFcn` —— 关闭窗口时响应函数；
- `KeyPressFcn` —— 键盘键按下时响应函数；
- `WindowButtonDownFcn` —— 鼠标键按下时响应函数；
- `WindowButtonMotionFcn` —— 鼠标移动时响应函数；
- `WindowButtonUpFcn` —— 鼠标键释放时响应函数；
- `CreateFcn` 和 `DeleteFcn` —— 建立和删除对象时响应函数；
- `CallBack` —— 对象被选中时回调函数等。

这些回调函数有的是针对窗口而言的，还有的是对具体控件而言的，学会了回调函数的编写将有助于高效编写 MATLAB 图形用户界面程序。

前面给出了窗口的常用属性，其实每个控件也有各种各样的属性，下面列出各个控件通用的常用属性。

- **`Units` 与 `Position` 属性** —— 其定义与窗口定义是一致的，这里就不加叙述了，但应该注意一点，这里的位置是针对该窗口左下角的，而不是针对屏幕的。
- **`String` 属性** —— 用来标注在该控件上的字符串，一般起说明作用或提示。
- **`CallBack` 属性** —— 此属性是图形界面设计中最重要的属性，它是连接程序界面整个程序系统的实质性功能的纽带。该属性值应该为一个可以直接求值的字符串，在该对象被选中和改变时，系统将自动地对字符串进行求值。一般地，在该对象被处理时，

经常调用一个函数，亦即回调函数。

- Enable 属性 —— 表示此控件的使能状态，如果设置为 'on' 则表示此控件可以选择，为 'off' 则表示不可选。
- CData 属性 —— 真色彩位图，为三维数组型，用于将真色彩图形标注到控件上，使得界面看起来更加形象和丰富多彩，该属性是 MATLAB 5.2 版本新引入的。
- TooltipString 属性 —— 提示信息显示，为字符串型。当鼠标指针位于此控件上时，不管是否按下鼠标键，都将显示提示信息。该属性也是 MATLAB 5.2 版本新引入的。
- Interruptable 属性 —— 可选择的值为 'on' 和 'off'，表示当前的回调函数在执行时是否允许中断，去执行其他的回调函数。
- 有关字体的属性，如 FontAngle, FontName 等。

利用 Guide 提供的强大功能，不但能设计一般的对话框界面，还可以设计更复杂的带有菜单的窗口，菜单系统的设置可以由 Guide 的菜单编辑器来完成。选择 Guide 的 Tools 菜单，则可以发现其中的 Menu Editor (菜单编辑器) 子菜单项，该菜单项将给出如图 2-35 所示的菜单编辑环境。

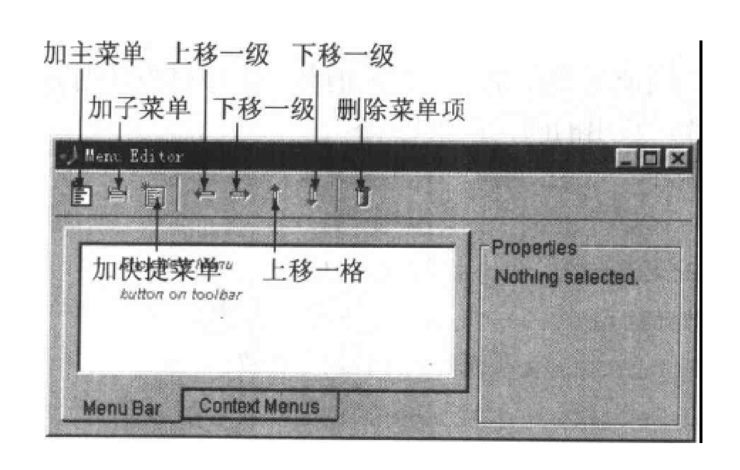


图 2-35 菜单编辑器界面

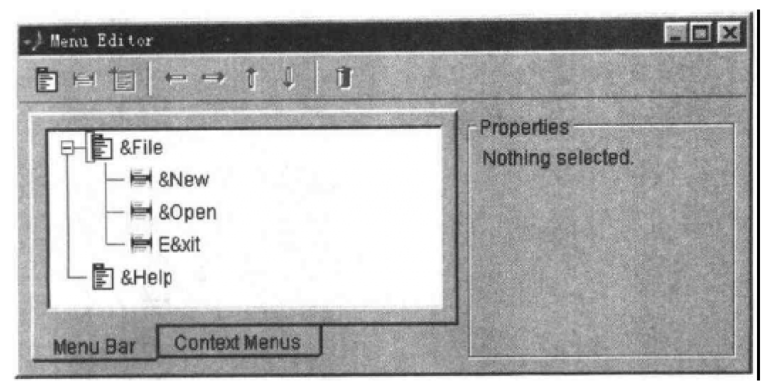
使用菜单编辑器，可以容易地按图 2-36 (a) 所示的格式编辑菜单，从而得出如图 2-36 (b) 所示的结果。

2.8.2 界面设计举例与技巧

本节将通过例子来演示 MATLAB 下图形用户界面设计的方法与思想，并介绍一些有关的编程技巧。

〖例 2.27〗 MATLAB 的图形界面设计实际上是一种面向对象的设计方法。假设想建立一个图形界面来显示和处理三维图形，最终图形界面的设想如图 2-37 所示。要求其基本功能是：

(1) 建立一个主坐标系，以备以后来绘制三维图形；







(a) 菜单编辑器编辑结果　　(b) 程序菜单

图 2-36　界面设计及修改

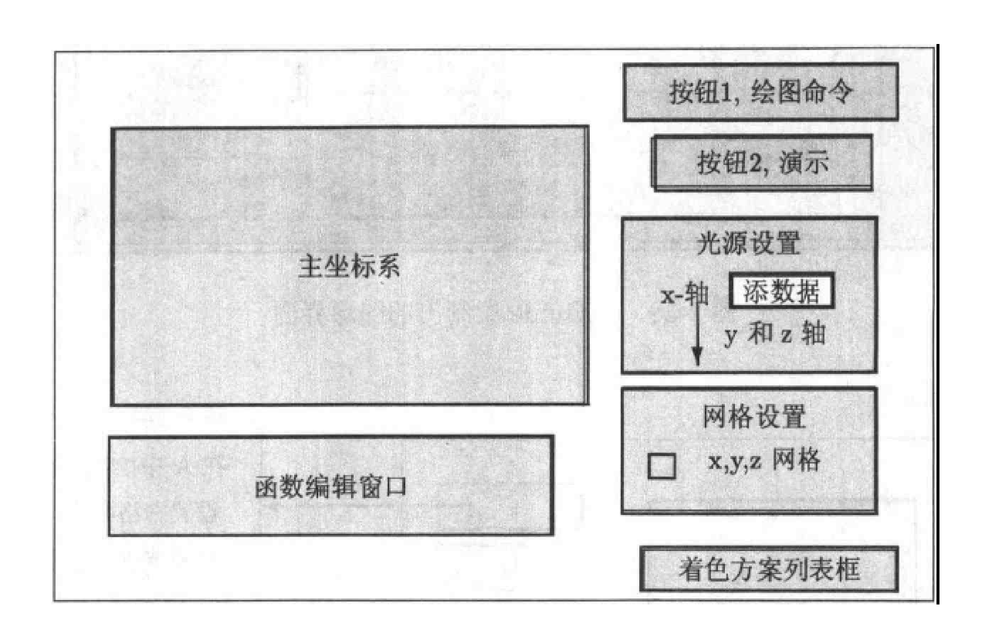


图 2-37　要建立的图形界面示意图

(2) 建立一个函数编辑框，接受用户输入的绘图数据；

(3) 建立两个按钮，一个用于启动绘图功能，另一个用于启动演示功能；

(4) 建立一组 3 个编辑框，用来设置光源在 3 个坐标轴的坐标值；

(5) 建立一组 3 个复选框，决定各个轴上是否需要网格；

(6) 建立一个列表框，允许用户选择不同的着色方法。

可以根据上面的设想，用 Guide 工具绘制出程序窗口的雏形，如图 2-38 所示。

根据上面的设想，可以把任务分配给各个控件对象，这就是面向对象的程序设计特点。其任务分配示意图如图 2-39 所示。从示意图中可以看出，A 和 B 两个部分并不承担任何的实际工作，它们只是给最终的绘图与数据编辑提供场所，所以它们的句柄是很有用的量。为了方便获得它们的句柄，分别将它们的标签 (即 Tag 属性) 设置为 myAxes 和 strB，同时为了使 strB 能接受多行的字符串输入，需要将其 Max 属性设置为大于 1 的数值，如取 100。

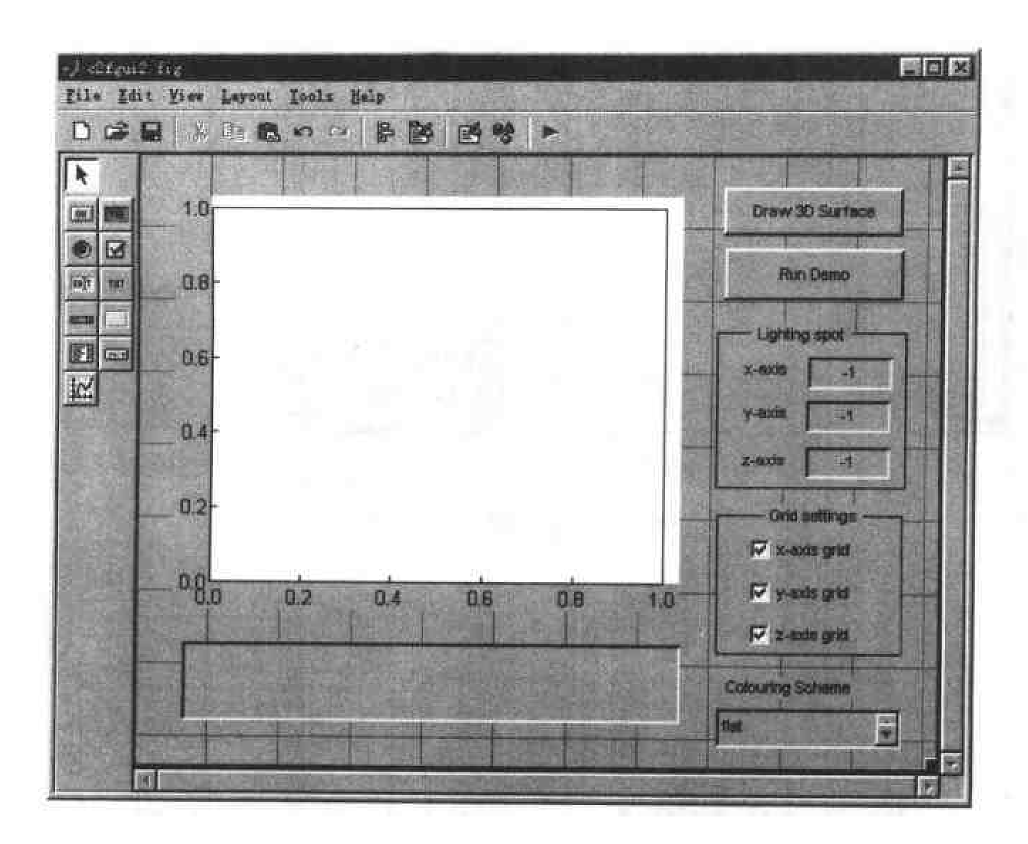


图 2-38　用 Guide 绘制出的图形界面

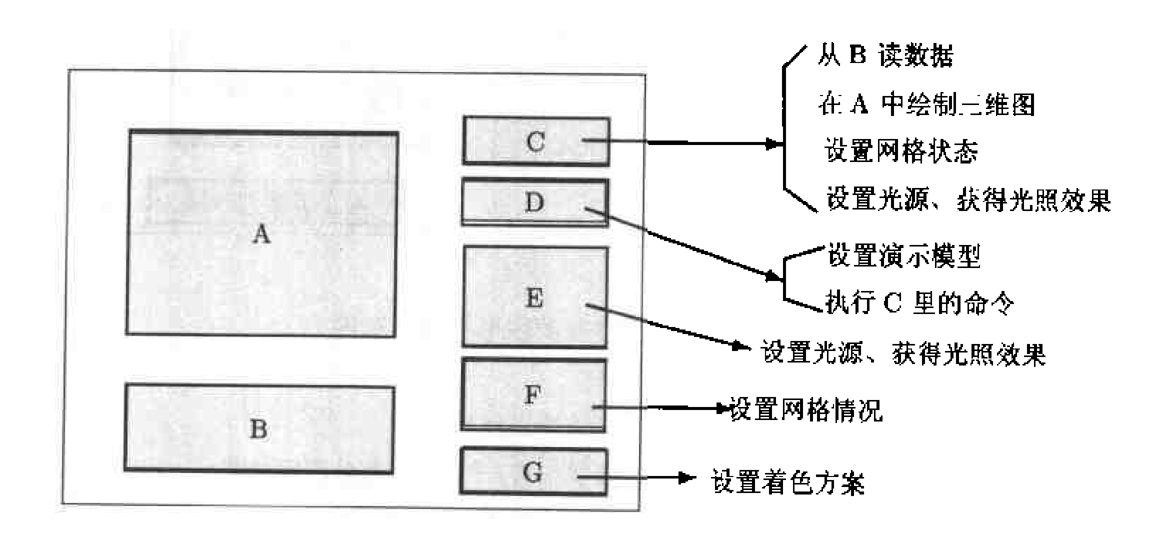


图 2-39　控件任务分配示意图

还可以将其他可能用到的控件标签分别设置为:

- C 区按钮的标签设置为 btnDraw;
- F 区三个网格检取框的标签分别 chkX, chkY 和 chkZ;
- E 区三个光照点坐标编辑框的标签分别为 edXPos, edYPos 和 edZPos;
- G 区着色方案列表框的标签设置为 lstColor。另外，单击 lstColor 的 String 属性左端的编

辑按钮，则可以在其中加上选项 flat | interp 等。

根据这里给出的任务分配图，可以创建出主程序界面，对应的函数名为 c2fgui2()，该函数的清单如下：

```
function varargout = c2fgui2(varargin)
if nargin == 0  % LAUNCH GUI
   fig = openfig(mfilename,'reuse');
   set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));
   handles = guihandles(fig);
   guidata(fig, handles);
   if nargout > 0
      varargout{1} = fig;
   end
elseif ischar(varargin{1})
   try
      if (nargout)
         [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
      else
         feval(varargin{:}); % FEVAL switchyard
      end
   catch
      disp(lasterr);
   end
end
```

可见，这样生成的主程序和前面生成的完全一致。那么两个不同的问题，界面描述上的差异在哪里呢？其实和早期版本不同，MATLAB 中描述界面的部分完全移到 fig 文件中了，主程序框架应该没有区别。另外，由于控件动作响应的不同，所以在编写事件响应子函数也是不同的。

根据任务分配中 C 区的要求，可以编写该按钮的回调函数，该函数从 strB 中读取字符串，然后在 myAxes 坐标系下将三维表面图绘制出来。这样就可以写出相应的子函数为：

```
function varargout = btnDraw_Callback(h, eventdata, handles, varargin)
try
   str=get(handles.strB,'String'); str0=[];
   for i=1:size(str,1) % 将所有输入的字符串串接起来
      str0=[str0, deblank(str(i,:))];
   end
   eval(str0); % 对字符串求值，
   axes(handles.myAxes); % 将坐标系设置为当前坐标系
   surf(x,y,z); % 绘制三维图
catch
```

```
    errordlg('数据有问题，请检查'); % 如果上述程序出错，则显示错误信息
end
```

注意，在该子函数中，使用了 try ··· catch 结构，这是为了防止函数编辑框中给出错误信息的，如果有错误信息，将弹出一个错误信息对话框。

现在再编写 D 区的回调函数，从其分配的任务来看，需要在 strB 编辑框中设置演示程序的数据赋值语句，然后再调用 btnDraw 的回调函数，所以可以写出如下的回调函数:

```
function varargout = btnDemo_Callback(h, eventdata, handles, varargin)
str1='[x,y]=meshgrid(-3:0.1:3, -2:0.1:2);';
str2='z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);'; % 写字符串的两行
set(handles.strB,'String',str2mat(str1,str2)); % 赋值
btnDraw_Callback(h, eventdata, handles, varargin); % 调用 btnDraw 的回调函数
```

下面看 E 区控件的回调函数如何编写，E 区有三个编辑框，分别放置光源点坐标的三个坐标轴位置，可以统一考虑这三个回调函数，分别从 edXPos, edYPos 和 edZPos 三个编辑框中读取数值，然后将 myAxes 坐标系下的图形进行光源设定。所以可以写出下面的回调函数:

```
function varargout = edXPos_Callback(h, eventdata, handles, varargin)
try
    xx=eval(get(handles.edXPos,'String')); % 读取光源位置
    yy=eval(get(handles.edYPos,'String'));
    zz=eval(get(handles.edZPos,'String'));
    axes(handles.myAxes); % 将坐标系设置为当前坐标系
    light('Position',[xx,yy,zz]); % 设置光源
catch
    errordlg('数据有问题，请检查'); % 如果上述程序出错，则显示错误信息
end
```

为简单起见，没有必要同时编写三个回调函数，只需按照图 2-40 所示的方式将 edXPos 的 Callback 栏目的内容复制到 edYPos 和 edZPos 的 Callback 栏目下即可。

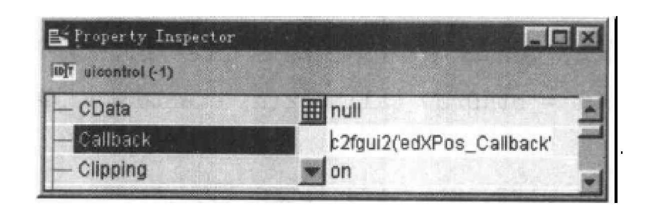


图 2-40 另外两个编辑框的 Callback 属性修改

类似地，可以编写 F 区的回调函数如下:

```
function varargout = chkX_Callback(h, eventdata, handles, varargin)
x=get(handles.chkX,'Value'); % 读取是否要网格标志
y=get(handles.chkY,'Value');
z=get(handles.chkZ,'Value');
```

```
set(handles.myAxes,'XGrid',onoff(x),'YGrid',onoff(y),'ZGrid',onoff(z))
function out=onoff(in) % 将 0,1 转换成 off, on
out='off'; if in==1, out='on'; end
```

该函数读取 chkX 等复选框的状态，并根据其结果设置网格的情况。因为这里不存在用户的字符串输入错误，故没有使用 try ··· catch 结构。在这里还编写了一个将 0,1 转换成字符串的 'off' 和 'on' 的子函数。要想正确执行这个程序，还需要用上面的方法手动修改 chkY 和 chkZ 的回调函数栏目，使之与 chkX 的完全一致。

最后应该编写 G 区的程序，该区要求从 lstColor 列表框中取出适当的选项，然后根据要求处理图形的着色。这样就能编写出如下的回调函数：

```
function varargout = lstColor_Callback(h, eventdata, handles, varargin)
v=get(handles.lstColor,'Value'); % 得出列表框的选项
axes(handles.myAxes); % 将坐标系设置为当前坐标系
switch v
case 1, shading flat; % 每块用同样颜色表示，无边界线
case 2, shading interp; % 插值平滑着色，无边界线
case {3,4}, shading faceted; % 带有黑色边界线
end
```

运行这样编写的程序 c2fgui2.m，可以得出如图 2-41 所示的界面表示，和文献 [54] 中给出的程序比较可以发现，程序大大地简化了，这是因为充分利用了 MATLAB 6.1 中的图形界面设计方法。

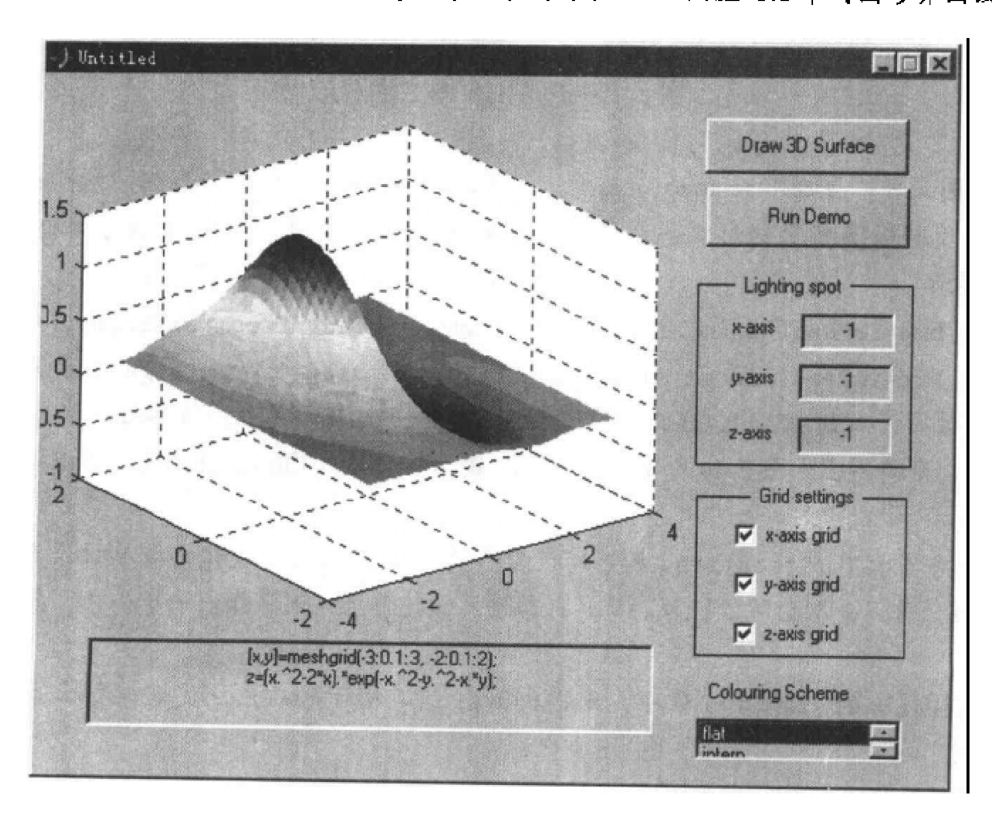


图 2-41 按下 Run Demo 按钮的效果

【例 2.28】 在实际程序的图形界面中，为了美观起见，经常需要用图形来表示按钮，考虑如图 2-42 所示的草图，从该图可以看出，该程序界面上需要一个坐标轴对象，用来显示二维曲线，将

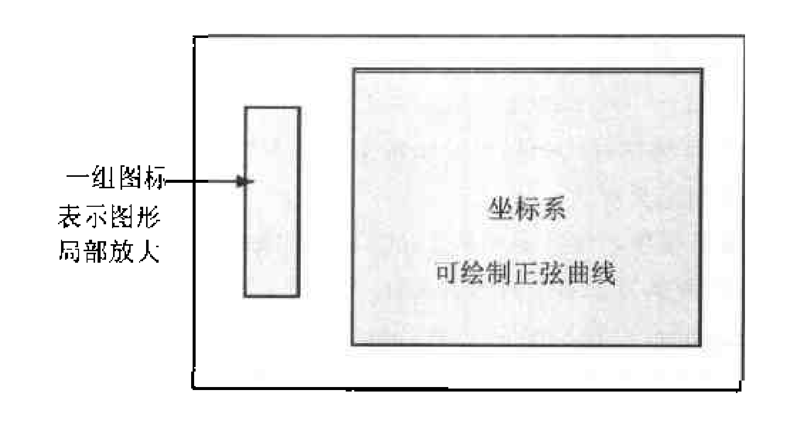


图 2-42 程序设计草图

其标签表示为 axPlot。另外，该程序需要 4 个按钮型对象，这 4 个按钮上均标注图形标记。

可以从成型的程序界面上取下有关的位图，并把它存成位图文件。这里将要采用一组表示局部放大的位图，如图 2-43 所示。

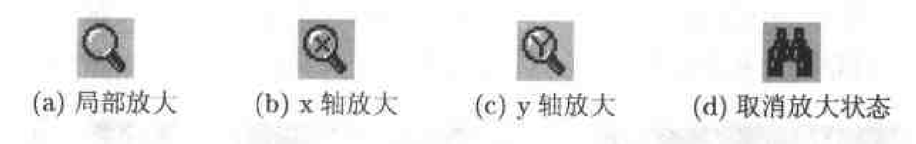


图 2-43 有关的位图文件示意

有了位图文件，再调用 imread() 函数，就可以将位图读成 MATLAB 可以接受的 $n \times m \times 3$ 的无符号 8 位整数多维数组形式，故可以给出以下的语句，可以用下面语句读出各个位图的 MATLAB 表示。

```
>> A_bmp=imread('zoom_sign.bmp'); D_bmp=imread('fzoom_sign.bmp');
   B_bmp=imread('zoomx_sign.bmp'); C_bmp=imread('zoomy_sign.bmp');
```

将这些位图读入到 MATLAB 工作空间后，可以打开按钮的属性编辑界面，单击 CData 属性右侧的方格，则将得出如图 2-44 所示的对话框，提示用户输入 CData 属性值，可以将 A_tmp 输入编辑框。

这里还将演示 TooltipString 属性的设置，将“局部放大”按钮的该属性设置为 '局部放大'，该属性的作用是在程序运行时，将鼠标移动到该控件上，则将弹出一个小条，显示该属性的信息。

运行这样得出的 c2fgui4.m 程序，则得出如图 2-45 所示的效果，注意该界面上的“局部放大”提示条。

可以编写下面的各个回调函数：

```
function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)
zoom on
```

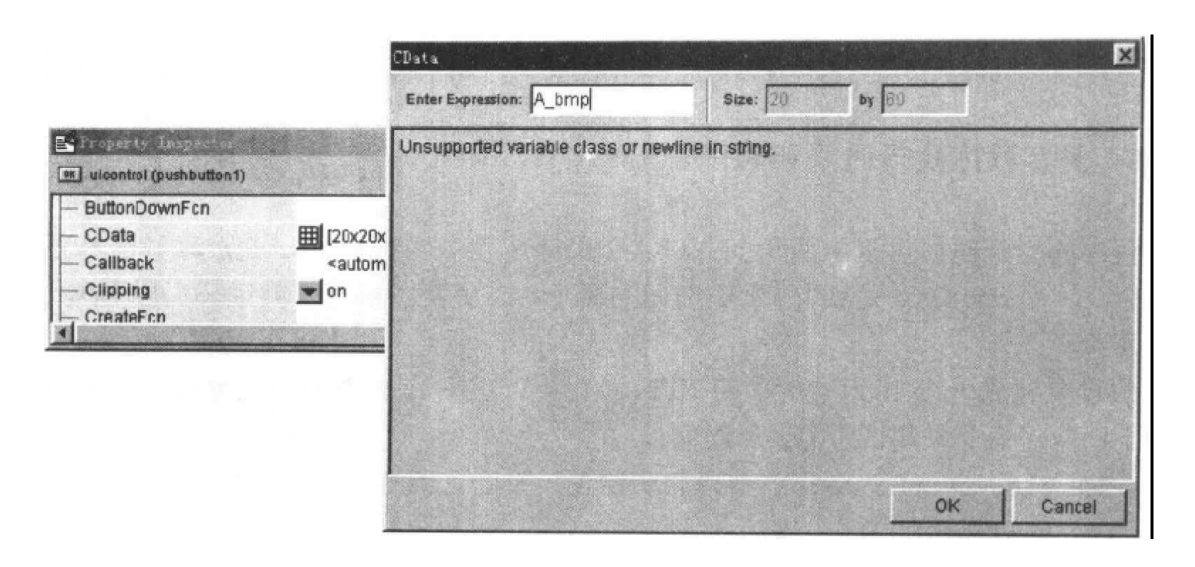


图 2-44　图形数据的输入

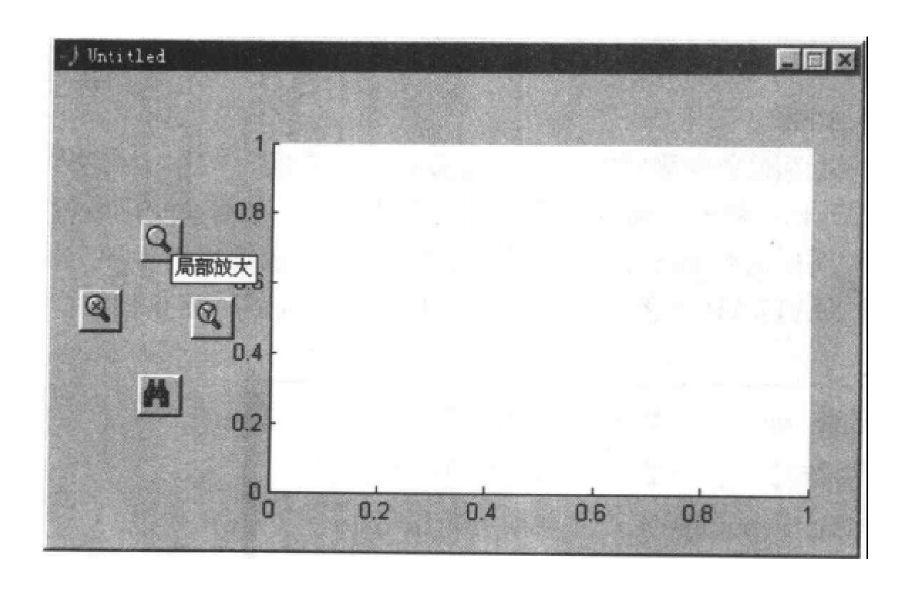


图 2-45　程序运行界面

```
function varargout = pushbutton2_Callback(h, eventdata, handles, varargin)
zoom xon
function varargout = pushbutton3_Callback(h, eventdata, handles, varargin)
zoom yon
function varargout = pushbutton4_Callback(h, eventdata, handles, varargin)
zoom off
```

更简单地，可以不编写回调函数，而只打开各个按钮的属性编辑界面，在该界面的 Callback 栏目下直接写出 'zoom on' 字样即可，就可以构造出 c2fgui5.m 程序，这样处理之后，程序将变得更简洁。

2.9　提高MATLAB执行效率的技巧

2.9.1　测定程序执行时间和时间分配

程序的运行时间可以由两组MATLAB命令来测取。tic, toc是启动秒表和停止秒表的命令，而cputime是获取CPU时间的命令，二者都可以用来测定某个程序执行所需的时间。

假设要建立一个1000×1000的Hilbert矩阵，并对之进行奇异值分解[1]，那么可以通过下面两对命令来测试所需要的时间。

```
>> tic, t=cputime; % 开始启动秒表，并记下当前的CPU时间
   A=hilb(1000);   % 运行程序
   toc, cputime-t  % 显示所需的时间
elapsed_time =
      0.6000
ans =
      0.6000
```

可以看出，这两对命令测定的时间十分接近，故在实际应用中用哪对命令都不会产生太大的误差，而tic和toc命令因为不产生附加变量，故而在实际编程中更常用。

M函数耗时剖析命令profile是MATLAB从5.0版开始提供的另一个实用功能。然而，不同版本MATLAB下的profile命令格式是不同的。在6.1版本中，profile命令的格式为：

```
profile on       % 启动耗时剖析功能
待测函数名        % 实际运行要测试的函数
profile report   % 产生耗时分析报告
profile off      % 结束剖析
```

测试完成后，将在临时目录中生成一个HTML文件，并启动一个Web浏览器调入该文件。用户可以从该浏览器下观察哪些语句耗时多，从而为改进程序提供某些依据。

下面剖析作者编写反馈系统分析与设计工具CtrlLAB的耗时状况

```
>> profile on;  ctrllab;  profile report
```

得出的耗时剖析结果在一个浏览器窗口中给出，如图2-46所示。每条语句的耗时都显示出来了，这样就可以从该窗口中发现哪些命令耗时多，从而对之进行有效修改，最终提高MATLAB程序的执行效率。

[1] 这个例子只用于演示测定时间的命令，因为这样大的Hilbert矩阵在数值上是没有任何意义的，因为十几阶的Hilbert矩阵在数值上就趋于奇异矩阵。

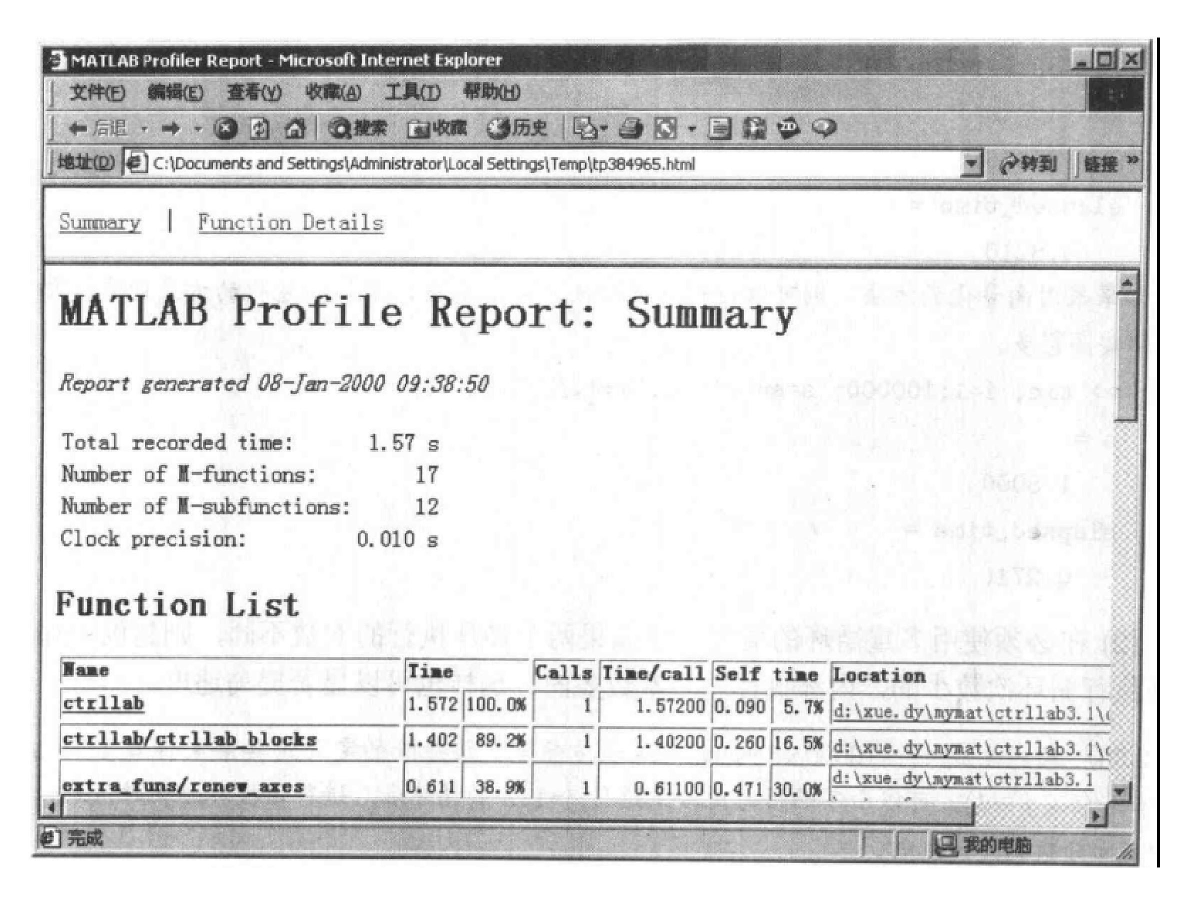


图 2-46 应用程序耗时剖析结果

2.9.2 加快 MATLAB 程序执行速度的建议

因为 MATLAB 语言是一种解释性语言，所以有时 MATLAB 程序的执行速度不是很理想。这里将依照作者十多年的实际编程经验给出加快 MATLAB 程序执行速度的一些建议和体会。

- **尽量避免使用循环**

循环语句及循环体经常被认为是 MATLAB 编程的瓶颈问题。改进这样的状况有两种方法：

(1) 尽量用向量化的运算来代替循环操作。这里将通过如下的例子来演示如何将一般的循环结构转换成向量化的语句。

〖例 2.29〗 *考虑下面无穷级数求和问题*

$$I=\sum_{n=1}^{\infty}\left(\frac{1}{2^n}+\frac{1}{3^n}\right)$$

如果只求出其中前有限项，比如 100,000 项之和[①]*，则可以采用下面的常规语句进行计算*

```
>> tic, s=0;
```

① 要精确地求出级数的和，无需求 100000 项，几十项往往就能得出满意的精度。这里主要是为了演示循环运算向量化的优越性。

```
for i=1:100000, s=s+(1/2^i+1/3^i); end, s, toc
s =
   1.5000
elapsed_time =
   0.8210
```

如果采用向量化的方法，则可以得出下面结果。可以看出，采取向量化的方法比常规循环运算效率要高得多。

```
>> tic, i=1:100000; s=sum(1./2.^i+1./3.^i), toc
s =
   1.5000
elapsed_time =
   0.2710
```

(2) 在必须使用多重循环的情况下，如果两个循环执行的次数不同，则建议在循环的外环执行循环次数少的，内环执行循环次数多的。这样也可以显著提高速度。

〖例 2.30〗 考虑生成一个 10000×5 的 Hilbert 长方矩阵，该矩阵的定义是其第 i 行第 j 列元素为 $h_{i,j} = 1/(i+j-1)$。可以由下面语句比较先进行 j=1:5 的循环和后进行该循环的耗时区别，其效果和前面分析的是一致的。

```
>> tic,
   for i=1:10000 % 大循环在外层
      for j=1:5
         H(i,j)=1/(i+j-1);
      end
   end
   toc
elapsed_time =
     23.6860
>> tic
   for j=1:5      % 大循环在内层
      for i=1:10000
         H1(i,j)=1/(i+j-1);
      end
   end
   toc
elapsed_time =
     4.0590
```

- **大型矩阵的预先定维**

给大型矩阵动态地定维是个很费时间的事。建议在定义大矩阵时，首先用 MATLAB

的内在函数，如 zeros() 或 ones() 对之先进行定维，然后再进行赋值处理，这样会显著减少所需的时间的。

再考虑例 2.30 中的问题，如果输入下面的命令

```
>> tic
   H=zeros(10000,5); % 大矩阵预定维
   for j=1:5
      for i=1:10000
         H(i,j)=1/(i+j-1);
      end
   end
   toc
elapsed_time =
     0.4210
```

则采用预先定维的方法，再结合向量化的方法，可以给出下面的 MATLAB 语句。

```
>> tic
   H=zeros(10000,5);
   for i=1:5
      H(:,i)=1./[i:i+9999]';
   end
   toc
elapsed_time =
     0.0010
```

可见，预先定维后，所需要的时间显著地减少了。可以看出，同样一个问题，由于采用了有效的措施，所需的时间就可以从 23.68 秒减少到 0.001 秒，亦即效率提高了 20000 多倍。

对二重循环这样的特殊问题，还可以使用 meshgrid() 函数构造两个 10000×5 矩阵 i 和 j，从而直接得出 H 矩阵，也可以得出较高的速度。

```
>> tic, [i,j]=meshgrid(1:10000,1:5); H=1./(i+j-1); toc
elapsed_time =
     0.1100
```

- **优先考虑内在函数**

矩阵运算应该尽量采用 MATLAB 的内在函数，因为内在函数是由更底层的编程语言 C 优化构造，并置于 MATLAB 内核中的，其执行速度显然快于使用循环的矩阵运算。

- **采用有效的算法**

在实际应用中，解决同样的数学问题经常有各种各样的算法。例如求解定积分的数值解法在 MATLAB 中就提供了两个函数 —— **quad()** 和 **quad8()**，其中后一个算法在精度、速度上都明显高于前一种方法[51]。所以说，在科学计算领域是存在"多快好省"的途径的。如果一个方法不能满足要求，可以尝试其他的方法。

• **应用 Mex 技术**

虽然采用了很多措施，但执行速度仍然很慢，比如说耗时的循环是不可避免的，这样就应该考虑用其他语言，如 C 或 Fortran 语言。按照 Mex 技术要求的格式编写相应部分的程序，然后通过编译连接，形成在 MATLAB 可以直接调用的动态连接库 (DLL) 文件，这样可以显著地加快运算速度。下面将介绍 Mex 格式程序的编写方法。

2.9.3 Mex 程序设计技术

为简便起见，MATLAB 中定义的数据结构在 C 语言中统一成一个类型：MATLAB 阵列型。这个阵列又不失其一般性，它既可以表示标量、向量和矩阵，又可以表示数据结构体与单元变量这样的新数据结构。在 C 语言中，可以用下面的命令来表示一个 MATLAB 变量 A：

```
mxArray *A;
```

字符串型变量还可以由 mxChar 定义。和数据类型有关的信息可以由下面的 C 语言函数直接测定。

• **检测一个输入变量的类型** 一个变量的类型可以调用下面的函数测出：

```
mxClassID k_id=mxGetClassID(mxArray *ptr)
```

此函数将检测指针 ptr 所指向的 mxArray 型变量的类型，返回的变量 k_id 为所测变量的类型，它是 Mex 下定义的 mxClassID 变量，C 语言中定义的表示 Mex 变量类型的常数在表 2-4 中给出。

表 2-4 MATLAB 支持的各种类标识表

类标识名	数据类型	类名	类标识名	数据类型	类名
mxDOUBLE_CLASS	双精度浮点	'double'	mxSINGLE_CLASS	单精度浮点	'single'
mxINT8_CLASS	8 位整型	'int8'	mxUINT8_CLASS	8 位无符号整型	'uint8'
mxINT16_CLASS	16 位整型	'int16'	mxUINT16_CLASS	16 位无符号整型	'uint16'
mxINT32_CLASS	32 位整型	'int32'	mxUINT32_CLASS	32 位无符号整型	'uint32'
mxCHAR_CLASS	字符型	'char'	mxSTRUCT_CLASS	数据结构体	'struct'
mxCELL_CLASS	单元数据	'cell'	mxUNKNOWN_CLASS	未知类	—

• **获得输入变量的元素总数** 指针 ptr 所指向的变量中元素总数可由下面函数测出：

```
int n=mxGetNumberOfElements(mxArray *ptr);
```

其中，n 即为该变量的元素总数，它的值相当于在 MATLAB 下对一个变量 A 使用函数 prod(size(A)) 得出的结果。

- **测出输入变量的维数** 指针 ptr 所指向变量的维数可以由下面的函数测出：

```
int m=mxGetNumberOfDimensions(mxArray *ptr);
```

其中，m 实际上是该多维数组的维数。该变量每一维的大小值可以由下面的函数测出：

```
int *ndims = mxGetDimensions(mxArray *ptr)
```

该函数返的 ndims 是一个整型的数组，其中 ndims[i] 为第 i+1 维的大小，这样该变量实际上是一个 ndims[0] × ndims[1] × ⋯ × ndims[m-1] 维的多维数组。

- **判定是否为某类变量** 例如可以由下面的函数测出：

```
bool k=mxIsChar(mxArray *ptr)
```

指针 ptr 所指向的变量是否为字符串，如果是则返回 1，否则返回 0，该返回变量为逻辑变量。同类的函数还有：mxIsCell(), mxIsClass(), mxIsNaN() 等，其涵义是很明显的，所以这里就不详细介绍了。

MATLAB 的早期版本中，Mex 格式下的 C 函数名很混乱，MATLAB 从 4.0 版本开始对函数名做了规范化处理，将有关的函数分为几类。当前版本中将函数按其功能分别用不同的前缀名标识：

- 前缀名 mx 主要涉及到建立和读取 MATLAB 变量的函数，一般是可以对 MATLAB 数组 (定义为 maArray) 进行直接操作，它们在 matrix.h 头文件中定义。
- 前缀名 mat 一般表示那些涉及到对 MAT 数据文件进行操作的函数，这类函数一般由 mat.h 头文件定义。
- 前缀名 mex 一般表示 C 对 MATLAB 函数功能的直接调用，这类函数一般由 mex.h 头文件定义。
- 前缀名 eng 表示涉及到 MATLAB 计算引擎的 C 函数，这类函数一般由 engine.h 头文件定义。

在构造的 C 语言文件的前面应当包含相应的头文件，因为上面的数据结构和各种函数都是在这些头文件中定义的。有了这样的定义，就可以在 C 下访问 MATLAB 的数据结构并执行相应的函数了。

当前的 MATLAB 可以使用以下的支持 32 位编程的 C 编译程序：

- Microsoft Visual C++ 5.0 及以上版本；
- Watcom C++ 10.6 版本；
- 免费的 LCC-win32[①]。

可以由 mex -setup 命令指定编译程序，并设定有关的编译环境

```
>> mex -setup
Please choose your compiler for building external interface (MEX)
files.  Would you like mex to locate installed compilers [y]/n?
```

① MATLAB 6.x 自带，或可以由下面网址免费下载：http://www.cs.virginia.edu/~lcc-win32。

当然，还可以通过手动选择的方式来给编译程序定位。但建议采用自动方式。通过自动搜寻，系统将给出如下的显示：

```
mex has detected the following compilers on your machine:
[1] : Microsoft 6.0 compiler in c:\Microsoft Visual Studio
[2] : Lcc compiler in c:\matlab6p1\sys\lcc
[3] : Watcom C 10.6 compiler in c:\watcom
[0] : None
Please select a compiler. This compiler will become the default:
Please verify your choices: 2
```

这时可以选择其中的第 2 选项，这样将得出如下的结果

```
Compiler: LCC 2.4
Location: C:\matlab6p1\sys\lcc
Are these correct?([y]/n):
The default options file:
"C:\WINDOWS\Application Data\MathWorks\MATLAB\mexopts.bat"
is being updated...
```

确认了之后，系统将自动设置好编译程序。以后用户只需键入 mex 命令就可以生成动态连接库文件了，该命令的结构为：

```
mex 选项 文件名
```

这里的文件名是字符串，应该包含后缀名。该命令既可以在 DOS 下调用，也可以在 MATLAB 命令窗口中调用。所有的选项均可以由 mex -h 命令列出，例如选项 -c 表示只编译，不形成 DLL 文件。如果不带有任何选项，则将自动生成同名的 DLL 文件。若想指定一个不同的 DLL 文件名，则可以使用 mex -output new_file 命令，这时将生成一个名为 new_file.dll 的文件。在 MATLAB 下可以直接调用该文件。一般在编译 Mex 函数之前，应该将该 *.c 文件复制到 MATLAB 当前的工作目录，如默认的 work 目录。

Mex 文件结构如图 2-47 所示。在此结构下，首先要调用一个 Mex 入口函数，然后获得输入变量的指针和变量内容，再执行 C 语言程序的主体部分，最后将结果写回到 MATLAB 环境下。Mex 文件的主函数入口语句应该由 mexFunction() 引导，该函数的调用格式是固定的：

```
void mexFunction( int nlhs, mxArray *plhs[],
    int nrhs, const mxArray *prhs[])
```

其中，nlhs[①] 和 nrhs 两个变量分别为该函数在 MATLAB 调用中的返回参数和输入参数的个数，分别相当于 MATLAB 中的 nargout 和 nargin。*plhs[] 和 *prhs[] 分别为返

[①] lhs 实际上是英文 left hand side 的缩写，由此可见，这些参数是有关函数调用格式中左边变量情况的信息，亦即返回信息；前面冠以 n 和 p 很显然是指变量个数和变量指针。同样地，rhs 是 right hand side，即指输入变量的情况。

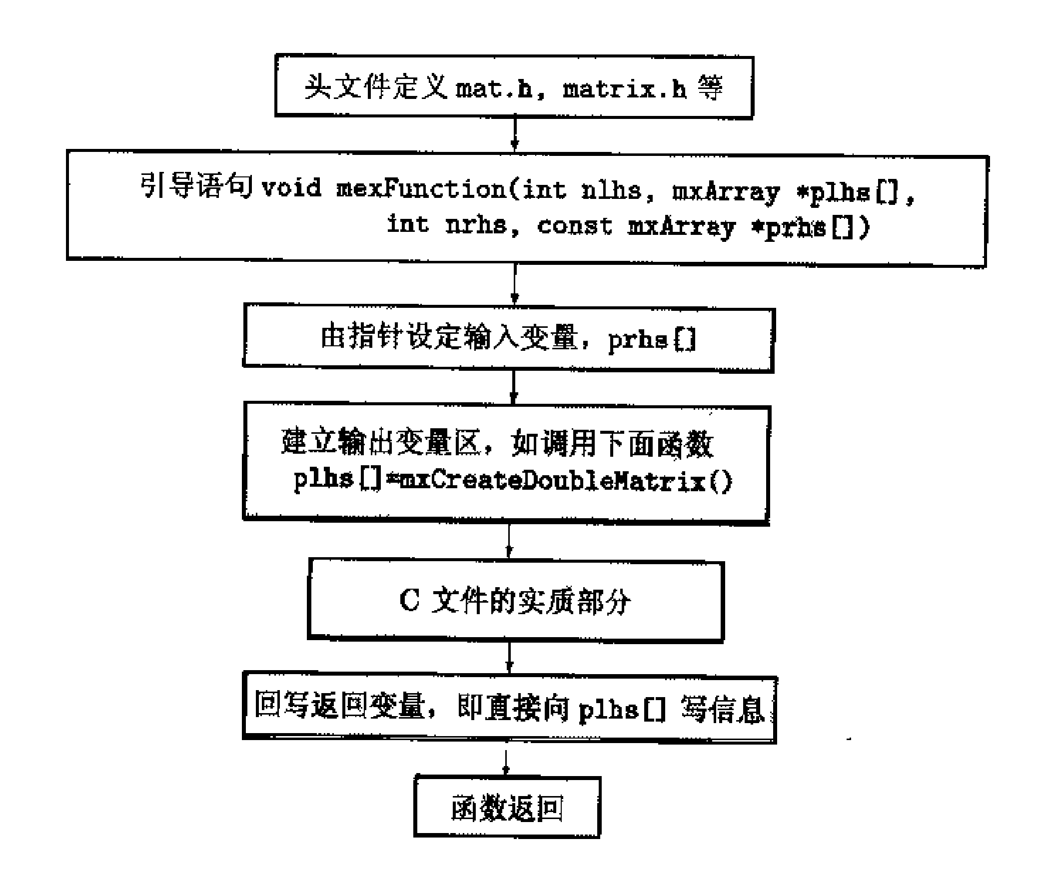


图 2-47 Mex 文件的基本结构

回参数和输入参数的指针，例如 *prhs[0] 为输入的第一个变量的指针，*prhs[1] 为输入的第二个变量指针等[②]。在函数调用时，nlhs、nrhs 和 *prhs[] 三个变量的指针均自动地确定，而 *plhs[] 需要动态地分配新的指针。下面的一些 Mex 格式下的 C 语言函数是在接口程序中经常使用的。

- **获得矩阵的行数和列数** 这两个功能可以分别由函数 mxGetM() 和 mxGetN() 来实现，比如说第 k 个输入变量的维数可以由 mxGetM(prhs[k-1]) 和 mxGetN(prhs[k-1]) 两个表达式求出。此函数实际上是 mxGetDimensions() 在矩阵问题上的简单表示形式。
- **获得矩阵变量的指针** 变量的指针可以由 mxGetPr() 函数得出，例如第 k 个输入变量的指针可以由 mxGetPr(prhs[k-1]) 获得。如果已经对第 k 个返回变量进行正确的定维，则可以使用 mxGetPr(plhs[k-1]) 来设定该返回变量的指针。

如果第 k 个变量为标量，则可以省去 mxGetM() 和 mxGetN() 函数的调用，而将 mxGetPr() 函数可以改为 mxGetScalar(prhs[k-1])。

值得指出的是，即使输入的变量是矩阵，在 C 下也应该由向量的形式来表示，而实际向量中的元素是将原矩阵中元素按列化成向量形式表示的结果。

- **判定一个矩阵是否为复数矩阵** 函数 mxIsComplex(prhs[k-1]) 可以判定第 k 个输

[②] 因为 C 语言数组下标从 0 开始，而 MATLAB 的数组下标从 1 开始，所以它们之间的值差 1。

入变量是否含有虚部，如果含有虚部，则此函数返回的结果为 1，否则为 0。复数矩阵的虚部矩阵指针可以由 mxGetPi(prhs[k-1]) 函数获得。

- **输出变量指针的动态分配** 矩阵输出可以用下面的函数分配指针：

```
plhs[k-1]=mxCreateDoubleMatrix(mrows,ncols, mxREAL);
```

其中 mrows 和 ncols 为新建矩阵的行数和列数；常数 mxREAL 为实数矩阵的类型，若使用 mxCOMPLEX 常数则表示要生成复数矩阵。调用了此函数后，系统将给第 k 个返回变量分配一个空间，该变量的实际指针可以用 mxGetPr() 设定。

在上面函数的调用中，均采用了 plhs 和 prhs 来设置变量的指针，而实际上，这些函数的调用并不限于这样的固定指针，而可以适用于任何 mxArray 型 MATLAB 数组。

【例 2.31】 假设想用 C 语言按 Mex 规则编写一个求矩阵乘积的函数，使得函数的调用格式为 C=mex_ex2_1(A,b)，那么可以写出如下的代码：

```
/*File name: mex_ex2_1*/
#include "matrix.h"
void mat_multiply(double *A, double *B, double *C,
   int mA, int nA, int mB, int nB)  % 矩阵相乘子程序
{
   int i,j,k,m=0;
   for (i=0; i<mA; i++){
      for (j=0; j<nB; j++){
         C[j*mA+i]=0;
         for (k=0; k<mB; k++){
            C[j*mA+i]+=A[k*mA+i]*B[j*mB+k];
      }}}
}
/* Main interface to MATLAB */
void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
{
   double *Ap, *Bp, *Cp;
   int mA,nA,mB,nB,mC,nC;
   Ap=mxGetPr(prhs[0]); Bp=mxGetPr(prhs[1]);  % 获得 A,B 矩阵指针
   mA=mxGetM(prhs[0]); nA=mxGetN(prhs[0]);    % 获得 A 矩阵的行列数
   mB=mxGetM(prhs[1]); nB=mxGetN(prhs[1]);    % 获得 B 矩阵的行列数
   plhs[0]=mxCreateDoubleMatrix(mA,nB,mxREAL);% 创建返回变量空间
   Cp=mxGetPr(plhs[0]);                       % 设置返回变量指针
   mat_multiply(Ap, Bp, Cp, mA, nA, mB, nB);  % 调用矩阵相乘子程序
}
```

其中，入口函数仍为 mexFunction()，在该函数中，首先从其输入的变量中获得 A 和 B 矩阵的指

针和维数，然后根据这两个矩阵的维数计算出结果矩阵的维数，并建立能存储这样一组数据空间的指针。然后可以调用 mat_multiply 函数进行矩阵乘法运算，得出的结果将自动写到 MATLAB 返回变量所在的指针处，从而完成整个调用过程。

可以在 MATLAB 提示符下输入 mex mex_ex2_1.c 命令，编译、连接此程序，形成动态连接库文件 mex_ex2_1.dll 文件。编译完成之后，可以用下面的 MATLAB 命令求解两个矩阵的乘积：

```
>> A=[1 2 3; 4 5 6]; B=[1 2; 3 4]; C=mex_ex2_1(A',B) % 矩阵可乘
C =
    13    18
    17    24
    21    30
>> D=A'*B  % 矩阵不可乘，得出错误结论
D =
    13    18
    17    24
    21    30
```

可见，这样得出的结果和由 MATLAB 得出的是完全一致的。另外，再运行下面的命令

```
>> A=[1 2 3; 4 5 6]; B=[1 2; 3 4]; C=mex_ex2_1(A,B)
C =
     7    10
    19    28
```

也能得出结果矩阵。观察 A 和 B 矩阵会发现，这两个矩阵实质上是不能相乘的，故所得出的结果是没有意义的。C 语言程序和 MATLAB 不同，在 C 下它只管设定指针，而不关心该指针所指数据是不是有物理意义。所以在用 C 语言实现某些功能时，还应该测试一下所采用的变量是否正确。如果给定的数据没有意义，那么还应该显示出报警信息。

MATLAB 下 Mex 程序生成的报警信息可以由 mexErrMsgTxt() 函数给出，该函数类似于 MATLAB 中的 error() 函数，不但能发出错误信息，还能中止程序的运行。加入输入输出变量检测和兼容性检测，则可以写出下面的程序：

```
/*File name: mex_ex2_1a*/
#include "matrix.h"
void mat_multiply(double *A, double *B, double *C,
   int mA, int nA, int mB, int nB)
{
   int i,j,k,m=0;
   for (i=0; i<mA; i++){
      for (j=0; j<nB; j++){
         C[j*mA+i]=0;
         for (k=0; k<mB; k++){
```

```
            C[j*mA+i]+=A[k*mA+i]*B[j*mB+k];
        }}}
    }
    /* Main interface to MATLAB */
    void mexFunction( int nlhs, mxArray *plhs[],
                      int nrhs, const mxArray *prhs[])
    {
       double *Ap, *Bp, *Cp;
       int mA,nA,mB,nB,mC,nC;
       if (nrhs!=2) mexErrMsgTxt("Wrong number of input arguments!");
       if (nlhs>1) mexErrMsgTxt("Too many output arguments!");
       Ap=mxGetPr(prhs[0]); Bp=mxGetPr(prhs[1]);
       mA=mxGetM(prhs[0]); nA=mxGetN(prhs[0]);
       mB=mxGetM(prhs[1]); nB=mxGetN(prhs[1]);
       if (nA!=mB) mexErrMsgTxt("Matrix dimensions not competible!");
       plhs[0]=mxCreateDoubleMatrix(mA,nB,mxREAL);
       Cp=mxGetPr(plhs[0]);
       mat_multiply(Ap, Bp, Cp, mA, nA, mB, nB);
    }
```

该函数进行编译连接后形成的 DLL 程序对前面的问题能得出正确结果。亦即，若给出的两个矩阵维数不相容，则能给出提示，并终止函数的运行。

其实，仔细研究此函数会发现它还是存在问题的：如果 A 和 B 两个矩阵有一个是标量，则程序仍然可能出错误；如果涉及到复数矩阵的运算，则此函数仍然是无能为力的，除非改变原来的程序。由此可以得出结论：用 C 语言编程相对于 MATLAB 编程需要考虑的烦琐事情要多得多，有一点小地方考虑不到，就将出现不可预见的结果。这也从另一个角度说明了 MATLAB 语言的优越性，因为程序设计者可以将精力集中在解决数学问题的算法上，而无需把时间花在这些无谓的繁琐小事上。

在比较正规的编程中，除了需要建立一个 DLL 文件之外，还应该在该目录下同时建立一个名为 mex_ex2_1a.m 的 M 函数，存放有关该函数说明的注释信息。例如，本例中可以建立一个帮助文件为

```
function mex_ex2_1a()
%MEX_EX2_1A is a help file for the DLL function MEX_EX2_1A.
```

这时，在该目录下将存在两个可以在 MATLAB 下执行的函数。在函数调用时，DLL 因其优先级别高而自动地执行，而在用 help 命令调用联机帮助时，系统将读出文件 mex_ex2_1a.m 中的帮助信息。

从上面的叙述中，读者基本可以了解 Mex 技术下程序编写的初步内容。可以通过下面的步骤来编写 Mex 规则下的 C 程序。

- mexFunction() 函数是整个 MATLAB 环境和 Mex 程序的接口，由该函数将输入列

表中各个变量的指针传递到 C 程序中。

- C 程序可以通过 Mex 提供的头文件定义，用 `mxGetPr()` 函数读取各个输入变量的指针，并由 `mxGetM()` 和 `mxGetN()` 两个函数读取输入变量的大小，这样就可以从内存中取出 MATLAB 的工作空间变量。
- 还可以通过 `mxCreateDoubleMatrix()` 函数给返回的变量开创内存空间，并用 `mxGetPr()` 函数设定指针，这样 C 程序的返回结果将能写到 MATLAB 环境可以读的位置。
- 程序编写完成后，执行 `mex` 命令将之变换成 DLL 文件。
- 编写一个可用于联机帮助的同名 M 文件。

这样就可以用像其他 MATLAB 本身的 M 函数那样的调用格式对之进行调用。

2.10 习 题

(1) 用 MATLAB 可以识别的格式输入下面两个矩阵

$$\boldsymbol{A}=\begin{bmatrix}1&2&3&3\\2&3&5&7\\1&3&5&7\\3&2&3&9\\1&8&9&4\end{bmatrix},\quad \boldsymbol{B}=\begin{bmatrix}1+4\mathrm{i}&4&3&6&7&8\\2&3&3&5&5&4+2\mathrm{i}\\2&6+7\mathrm{i}&5&3&4&2\\1&8&9&5&4&3\end{bmatrix}$$

再求出它们的乘积矩阵 $\boldsymbol{C}$，并将 $\boldsymbol{C}$ 矩阵的右下角 2×3 子矩阵赋给 $\boldsymbol{D}$ 矩阵。赋值完成之后，调用相应的命令查看 MATLAB 工作空间的占用情况。

(2) 解线性方程

$$\begin{bmatrix}5&7&6&5&1\\7&10&8&7&2\\6&8&10&9&3\\5&7&9&10&4\\1&2&3&4&5\end{bmatrix}\boldsymbol{X}=\begin{bmatrix}24&96\\34&136\\36&144\\35&140\\15&60\end{bmatrix}$$

(3) 考虑第 (1) 题中的 $\boldsymbol{A}$ 矩阵，如果再给出 `A(5,6)=3` 命令将得出什么结果，试理解该赋值方式。

(4) 试用简单的语句输入下面的 Jordan 矩阵

$$\boldsymbol{A}=\begin{bmatrix}\boldsymbol{A}_{11}&0&0\\0&\boldsymbol{A}_{22}&0\\0&0&\boldsymbol{A}_{33}\end{bmatrix}$$

其中

$$\boldsymbol{A}_{11}=\begin{bmatrix}-2&1&0\\0&-2&1\\0&0&-2\end{bmatrix},\quad \boldsymbol{A}_{22}=\begin{bmatrix}-3&1&0\\0&-3&1\\0&0&-3\end{bmatrix},\quad \boldsymbol{A}_{33}=\begin{bmatrix}-4&1&0\\0&-4&1\\0&0&-4\end{bmatrix}$$

(5) 用 MATLAB 语言实现下面的分段函数

$$y=f(x)=\begin{cases} h, & x>D \\ h/Dx, & |x|\leq D \\ -h, & x<-D \end{cases}$$

(6) 测试一下：若 `A=[1 2 NaN Inf -Inf 5 NaN]`，则运行后面各函数将返回什么结果，并解释各个结果的原因：`isnan(A)`, `isfinite(A)`, `isinf(A)`, `any(A)` 或 `all(A)`？

(7) 分别用 `for` 和 `while` 循环结构编写程序，求出

$$K=\sum_{i=0}^{63}2^i=1+2+2^2+2^3+\cdots+2^{62}+2^{63}$$

并考虑一种避免循环的简洁方法来进行求和，并比较各种算法的运行时间。

(8) 用循环语句形成一个有 20 个分量的数组，使其元素满足 Fibonacci 规则，即令数组的第 $k+2$ 个元素满足 $a_{k+2}=a_k+a_{k+1}$, $k=1,2,\cdots$，且 $a_1=1$, $a_2=1$。

(9) 用 MATLAB 语言编写一个函数来实现下面介绍的一元方程求解算法——二分法。假设有一个一元方程为 $f(x)=0$，那么我们的最终目的是求取它在 $[a,b]$ 区间内的实数解。前提条件是 $f(a)f(b)<0$。这样就保证了方程在这个区间内至少有一个实数根。

注 从解法上可以令 $x_1=a$, $x_2=b$。二分法的基本思想是，首先取这个区间的中点 $x_m=(a+b)/2$，判定 $f(x_1)$ 和 $f(x_2)$ 二者中哪一个和 $f(x_m)$ 异号。找到了该点后，问题转换成由该点和 x_m 点区间上的求解问题。重复这样的步骤，直到区间的长度小于一个可以接受的小正数 ϵ，则认为中点是原方程的解。

(10) 试用不同的方法展开多项式

$$P(s)=(s^2+1)^3(s+5)^2(s^4+4s^2+7)$$

并比较其结果。

(11) 选择合适的步距绘制出下面的图形

① $\sin(1/t)$，其中 $t\in(-1,1)$

② $\sin(\tan t)-\tan(\sin t)$，其中 $t\in(-\pi,\pi)$。

提示 图形中有些地方的函数值变化较大，而另一些地方变化较小。可以重新设定自变量向量，例如可以采用变步长方法以绘制出更高精度的曲线。

(12) 试同时绘制出正弦曲线，并用 MATLAB 中提供的图形编辑工具将其变成绿色的实线，且设置其宽度为 7，并试着将纵轴的正方向反向（即从上到下）。

(13) 试在图形坐标系下用 TEX 格式 $\int_0^{\Gamma(\alpha)}\Theta[x,\beta(t)]\mathrm{d}x$ 写出字符串，注意，按照标准的科学排版格式，公式中的字体为斜体，而微分运算符 d 为正体，试感受 `teximage_c()` 函数的效果。

(14) 对合适的 θ 范围选取分别绘制出下列极坐标图形：

① $\rho=\cos(7\theta/2)$　　② $\rho=1-\cos^3(7\theta)$

(15) 用图解的方式找到下面两个方程构成的联立方程的近似解

$$x^2+y^2=3xy^2,\ x^3-x^2=y^2-y$$

(16) 试绘制出

$$z=f(x,y)=\frac{1}{\sqrt{(1-x)^2+y^2}}+\frac{1}{\sqrt{(1+x)^2+y^2}}$$

的三维图。

(17) 三维图形还可以用伪色彩 (pseudo colour) 图形的形式表现出来，所使用的函数为 pcolor()，试用伪色彩的形式绘制出上面的图形。

(18) 在三维图形中，如果 Z 变量中的某个数值为 NaN，则在绘图中将忽略该点，所以在绘图时经常有意地将某个区域的 Z 值设置为 NaN，以便能剪下相应的部分，试利用此技术将上面绘制的三维图中心部分 $x^2+y^2<0.5$ 的圆剪去。

(19) 将一幅图片文件在 MATLAB 图形窗口上显示出来，并试着将其旋转 90°。另外，观察图片数据的三维数组，它们的值将为 0-255 之间的无符号 8 位整数，进行怎样的简单处理能除去图片中的蓝色分量？

(20) 创建一个新的图形窗口，使之背景颜色为绿色，并在窗口上保留原有的菜单项。假设想在鼠标器的左键按下之后在命令窗口上显示出 Left Mouse Button Pressed 字样的信息，试用 M 函数的形式实现这样的功能。

(21) 试用 Mex 格式编写一个 C 语言程序，使之能生成 10000×5 的 Hilbert 矩阵，并与前面介绍的纯 MATLAB 比较程序执行的速度。

(22) 设计一个数字电子表型的图形界面，并使之能自动地实时显示时间。

第 3 章 MATLAB 语言在现代科学运算中的应用

系统仿真技术本身涉及了大量的数学运算，计算机有巨大的求解数学运算的潜力，其求解问题的能力主要取决于算法与算法的代码实现。MATLAB 是一种高效的、高精度的科学运算语言，用它可以轻易地求解看似很复杂的数学问题，且较容易上手。

在本章中，将用较大的篇幅全面介绍数值计算技术及 MATLAB 求解方法，并探讨在若干问题中解析解方法。第 3.1 节中主要探讨数学问题数值解法的意义和必要性，第 3.2 节侧重讨论线性代数问题的数值解法，包括特殊矩阵、矩阵的参数求解、矩阵的相似变换与分解、矩阵的特征值问题、矩阵求逆、矩阵的非线性运算等，用 MATLAB 求解的方法将是很直观、方便且可靠的。最后将介绍依赖 MATLAB 的符号运算工具箱在求解线性代数问题解析解中的应用。第 3.3 节中将介绍微积分问题的求解方法，将分别介绍数值差分和微分算法、一般的数值积分算法和多重积分问题的计算机求解，最后将给出微积分及相关问题的解析解法。第 3.4 节将讨论动态系统仿真领域的数学基础 —— 常微分方程的数值解法，将首先介绍常用的数值算法，然后通过例子演示一般微分方程、隐式微分方程、刚性微分方程、 微分代数方程等问题的求解，最终还将探讨微分方程组的变换方法、微分方程解析解法问题和二阶系统的两点边值问题的求解方法与实例。第 3.5 节将着重讨论最优化问题的数值解法，首先介绍基于最优化技术的非线性方程组的求解方法，然后分别介绍无约束最优化问题、线性规划问题、二次型规划问题及一般非线性规划问题的数值解法。第 3.6 节将介绍数据处理的一般方法，如一维与二维插值问题、数据的最小二乘拟合 技术、简单的数据排序方法、伪随机数生成方法、数据分析与统计处理以及快速 Fourier 变换技术及其应用，还将介绍信号的相关函数计算和功率谱密度计算。本章的一些内容来源于文献 [54]，针对 MATLAB 新版本的特色和仿真技术的具体应用作了大幅度的扩充。无疑，本章介绍的内容将是基于 MATLAB/Simulink 进行数值仿真的数学基础。

3.1 解析解与数值解

现代科学与工程的进展离不开数学。数学家们感兴趣的问题和其他科学家、工程技术人员所关注的问题是不同的。数学家往往对数学问题的解析解，或称闭式解 (closed-form solution) 和解的存在性严格证明感兴趣，而工程技术人员一般对如何求出数学问题的解更关心，换句话说，能用某种方法获得问题的解则是工程技术人员更关心的问题。而获得这样解的最直接方法就是通过数值解法技术。在实际应用中，至少有两种情况需要数值解法：

- **解析解不存在时**

解析解不存在的情况在数学上并不罕见，甚至可以说，这样的现象是常见的。例如定积分 $\int_a^b e^{-x^2/2}dx$ 在上下限均为有穷时就没有解析解，虽然数学家用其他的符号去定义这样的解，但解的值到底多大却不是一目了然的。所以，在这样的情况下，要想获得积分的值，就必须采用数值解技术。

再例如，圆周率 π 的值本身就没有解析解，中国古代的数学家、天文学家祖冲之早在公元 480 年就算定了该值在 3.1415926 和 3.1415927 之间，但直到现在仍有"数学家"在试图计算出更多的有效位，甚至 1995 年 10 月，有人算出 6442450938 位来，但可以说，即使是这样的值也不是解析解！实际上，在一般科学与工程应用中，没有必要取那么多位的解，取 60 多亿位本身要占用多少计算机存储空间啊！在一般应用中取到祖冲之得到的近似值足矣，再精确的运算也至多取 20 多位就足够了。对计算机来说，取再多的位非但不会改进结果的精度，反而会加大计算机的负担，造成不必要的损失和浪费。其实，对问题的估算来说，使用公元前 250 年 (?) 阿基米德的 3.1418 也未尝不可，没有必要非去追求不存在的解析解不可。所以在这样的问题上，数值解法的优势就显示出来了。

- **解析解存在但不实用时**

例如，考虑 n 元一次代数方程组的求解问题，由著名的 Cramer 法则，可以把该问题化简为 n 个 $n-1$ 元一次方程组的求解，而每个 $n-1$ 元方程组的解又可以简化为 $n-1$ 个 $n-2$ 元一次方程组进行求解。从理论上讲，总可以把多元一次的方程组简化成解析可解的形式，从而可以得出结论：n 元一次方程组的解析解是可以求出来的。

不幸的是，n 元一次方程组的求解需要进行 $(n-1)(n+1)!+n$ 次基本运算。如果想求解不是很大规模的 20 元方程问题，需要进行 9.7073×10^{20} 次基本运算，即使用当今世界上速度最快的每秒 20 万亿次巨型计算机去求解这样的问题，也需要解上 2 年，使用"银河"类巨型机得计算数千年！而在某些科学工程计算中，又常常需要求解成百上千个变元的问题，指望用传统的方法解析解的方法来求解也就成了天方夜谭。

如果不去追求解析解，而想得到数值解，则可以通过计算数学的成果将原方程进行变换，比如用 Gauss 消元法等数值方法，这样 550 个变元的方程在一般个人计算机上用 MATLAB 在 1 秒钟内就可以解决问题，而求解上千个变元的问题也用不了多长的时间。

数学问题的数值解法已经成功地应用于各个领域。例如，在力学领域，常用有限元法求解偏微分方程；在航空、航天与自动控制领域，经常用到数值线性代数与常微分方程的数值解法等解决实际问题；在工程与非工程系统的计算机仿真中，核心问题的求解也需要用到各种差分方程、常微分方程的数值解法；在高科技的数字信号处理领域，离散的快速 Fourier 变换 (FFT) 已经成为其不可或缺的工具。在科学工程研究中能掌握一个或多个实用的计算工具，无疑会为研究者提供解决实际问题的强有力手段。

虽然 MATLAB 语言可以求解几乎所有的数值问题，但本章中只着重介绍在系统仿真领域常见的问题，如数值线性代数问题、数值微分与数值积分问题、常微分方程的数值解法、 非线性方程求解与最优化问题、数据插值与统计分析问题等。

3.2 数值线性代数问题及求解

3.2.1 特殊矩阵的 MATLAB 输入

在开始介绍矩阵运算之前，有必要先介绍一些特殊矩阵的定义及其 MATLAB 实现，这里要遇到的很多运算可以由 MATLAB 直接完成。

- **零矩阵、幺矩阵和单位矩阵**

在一般的矩阵理论中，把所有元素都为零的矩阵定义成零矩阵，把元素全为 1 的矩阵称为幺矩阵，把主对角线元素均为 1，而其他元素全部为 0 的方阵称为单位矩阵。这里进一步扩展单位阵的定义，使其为 $m\times n$ 的矩阵。零矩阵、幺矩阵和扩展单位阵的 MATLAB 生成函数分别为 `A=zeros(m,n)`, `ones(m,n)` 和 `A=eye(m,n)`，其中 `m` 和 `n` 分别为矩阵 $\boldsymbol{A}$ 的行数和列数。如果使用了命令 `A=zeros(n)`, `ones(n)` 或 `A=eye(n)`，则将分别产生 $n\times n$ 的零矩阵、幺矩阵和单位矩阵。如果 $\boldsymbol{B}$ 是一个给定的矩阵，则在 MATLAB 中可以用 `A=zeros(size(B))` 来定义一个和 $\boldsymbol{B}$ 阵同样大小的零矩阵。

函数 `zeros()` 和 `ones()` 还可用于多维数组的生成，例如，`zeros(3,4,5)` 将生成一个 3×4×5 的三维数组，其元素全部为 0。

- **随机元素矩阵**

顾名思义，随机元素矩阵的各个元素是随机产生的。如果矩阵的随机元素满足 $[0,1]$ 区间上的均匀分布，则可以由 MATLAB 函数 `rand()` 来生成，该函数通常的调用格式为 `A=rand(n,m)`。函数 `rand()` 还可以用于多维数组的生成。满足标准正态分布的随机数矩阵可以由 `randn()` 函数获得。

这里的随机数实际上是“伪随机数”，所谓伪随机数，就是通过某种数学公式生成的、满足某些随机指标的数据。这样的随机数是可以重复的，与某些用电子方法获得的不可重复的随机数是不同的。后面将更详细介绍有关伪随机数生成与性质。

- **对角矩阵**

对角矩阵是一种特殊的矩阵，这种矩阵的主对角线元素可以为零或非零元素，而非对角线元素的值均为 0。对角矩阵的数学描述方法为 $\mathrm{diag}(\alpha_1,\alpha_2,\cdots,\alpha_n)$，其中对角矩阵的矩阵表示为：

$$\mathrm{diag}(\alpha_1,\alpha_2,\cdots,\alpha_n)=\begin{bmatrix}\alpha_1 & & & \\ & \alpha_2 & & \\ & & \ddots & \\ & & & \alpha_n\end{bmatrix} \tag{3.1}$$

如果用 MATLAB 提供的方法建立一个向量 $\boldsymbol{V}=[\alpha_1,\alpha_2,\cdots,\alpha_n]$，则对角矩阵可以用 MATLAB 函数 `diag(V)` 来建立，其对角线上各个元素由向量 `V` 给出。

如果给定的 `V` 为一个矩阵，则 `diag(V)` 将提取出该矩阵的对角元素构成的向量。

- ***Hilbert* 及逆 *Hilbert* 矩阵**

Hilbert 矩阵是一类特殊矩阵，它的第 (i,j) 元素的值满足 $h_{i,j}=1/(i+j-1)$，这时

一个 $n\times n$ 阶的 Hilbert 矩阵可以写成：

$$\boldsymbol{H}=\begin{bmatrix}1 & 1/2 & 1/3 & \cdots & 1/n\\ 1/2 & 1/3 & 1/4 & \cdots & 1/(n+1)\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1/n & 1/(n+1) & 1/(n+2) & \cdots & 1/(2n-1)\end{bmatrix} \tag{3.2}$$

产生 Hilbert 矩阵的 MATLAB 函数 `A=hilb(n)`，其中 `n` 为要产生的矩阵阶次。

高阶 Hilbert 矩阵一般为坏条件的矩阵，所以直接对之求逆一般往往会引出浮点溢出的现象。MATLAB 提供了直接求取逆 Hilbert 矩阵的算法及函数，其调用方法为 `B=invhilb(n)`。

- **伴随矩阵**

假设有一个首一化的多项式

$$P(s)=s^n+a_1s^{n-1}+a_2s^{n-2}+\cdots+a_{n-1}s+a_n \tag{3.3}$$

则可以写出一个伴随矩阵：

$$\boldsymbol{A}_c=\begin{bmatrix}-a_1 & -a_2 & \cdots & -a_{n-1} & -a_n\\ 1 & 0 & \cdots & 0 & 0\\ 0 & 1 & \cdots & 0 & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ 0 & 0 & \cdots & 1 & 0\end{bmatrix} \tag{3.4}$$

生成伴随矩阵的 MATLAB 函数调用格式为 `B=compan(p)`，其中 `p` 为一个多项式向量，该函数将自动对多项式进行首一化处理。

- **Hankel 矩阵**

假设有一个序列 $\boldsymbol{C}$，其各个元素为 $\{c_1,c_2,\cdots,c_n,\cdots\}$，则可以写出一个矩阵，其第 (i,j) 元素满足 $h_{i,j}=c_{i+j-1}$，$i,j=1,2,\cdots,n$。这样可以构造一个矩阵：

$$\boldsymbol{H}=\begin{bmatrix}c_1 & c_2 & \cdots & c_n\\ c_2 & c_3 & \cdots & c_{n+1}\\ \vdots & \vdots & \ddots & \vdots\\ c_n & c_{n+1} & \cdots & c_{2n-1}\end{bmatrix} \tag{3.5}$$

这样的矩阵称为Hankel矩阵。如果 $n\to\infty$，则可以构造无穷型Hankel矩阵。Hankel矩阵是对称矩阵，且其反对角线上所有的元素都相同。Hilbert 矩阵是一种特殊的Hankel矩阵。

在 MATLAB 语言中，如果已知一个向量 $\boldsymbol{C}$，则可以由 `hankel(C)` 函数来构造出一个 Hankel 矩阵。例如取 `C=[1,2,3]`，则可以构造一个 3 阶的 Hankel 矩阵。

```
>> C=[1,2,3]; H=hankel(C)
H =

     1     2     3
```

```
        2     3     0
        3     0     0
```

可见，如果直接用单个向量来生成 Hankel 矩阵，则会自动地将该向量的各个元素填写到矩阵的第一列中去，然后利用其反对角线元素的值相等这一特点写出其他的元素，而主反对角线下的各个元素均设置为 0。MATLAB 还提供了该函数的另外一种调用方法：给定两个向量 C 和 R，如果用 H=hankel(C, R) 来生成 H，则首先将 H 矩阵的第一列的各个元素定义为 C 向量，将最后一行各个元素定义为 R，这样就可以依照 Hankel 矩阵反对角线上元素相等这一特性来写出相应的 Hankel 矩阵。根据 Hankel 矩阵的性质，其最后一行的第一个元素应该等于第 1 列的最后一个元素，如果冲突将给出元素冲突的警告信息。例如，若 C=[1,2,3], R=[3,9,10]，则将得出下面的结果：

```
>> C=[1,2,3]; R=[3,9,10];  H = hankel(C, R)
H =

        1     2     3
        2     3     9
        3     9    10
```

- **Vandermonde** 矩阵

假设有一个序列 $\boldsymbol{C}$，其各个元素满足 $\{c_1, c_2, \cdots, c_n\}$，则可以写出一个矩阵，其第 (i,j) 元素满足 $v_{i,j} = c_i^{n-j}$, $i,j = 1,2,\cdots,n$。这样可以构成一个矩阵：

$$V = \begin{bmatrix} c_1^{n-1} & c_1^{n-2} & \cdots & c_1 & 1 \\ c_2^{n-1} & c_2^{n-2} & \cdots & c_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_n^{n-1} & c_n^{n-2} & \cdots & c_n & 1 \end{bmatrix} \tag{3.6}$$

该矩阵称作 Vandermonde 矩阵。如果已知一个向量 $\boldsymbol{C}$，则可以由 MATLAB 提供的 V=vander(C) 函数来构造一个 Vandermonde 矩阵。例如若向量 C=[1,2,3,4,5]，则可以得出该向量对应的 Vandermonde 矩阵为：

```
>> C=[1, 2, 3, 4, 5];  V=vander(C)
V =

        1     1     1     1     1
       16     8     4     2     1
       81    27     9     3     1
      256    64    16     4     1
      625   125    25     5     1
```

3.2.2 矩阵的特征参数运算

MATLAB 提供了大量的矩阵特征参数求取函数，在这里首先叙述各个矩阵特征参数的意义和求解方法，然后介绍各个参数的 MATLAB 求解方法。

- **矩阵的行列式** (determinant)

矩阵 $\boldsymbol{A}=\{a_{ij}\}$ 的行列式定义为

$$\boldsymbol{D}=|A|=\det(A)=\sum(-1)^k a_{1k_1}a_{2k_2}\cdots a_{nk_n} \tag{3.7}$$

式中 $k_1,k_2,\cdots,k_n$ 是将序列 $1,2,\cdots,n$ 的元素交换 k 次所得出的一个序列，每个这样的序列称为一个置换 (permutation)；而 Σ 表示对 $k_1,k_2,\cdots,k_n$ 取遍 $1,2,\cdots,n$ 的所有排列的求和。

计算矩阵的行列式有多种算法，在 MATLAB 中采用的方法是对原矩阵 $\boldsymbol{A}$ 进行三角分解(又称为 LU 分解，后面将介绍)，将其分解成一个上三角矩阵 $\boldsymbol{U}$ 和一个下三角矩阵 $\boldsymbol{L}$ 的积，即 $\boldsymbol{A}=\boldsymbol{LU}$，这样可以先求出 $\boldsymbol{L}$ 矩阵的行列式。注意，在这一矩阵中只有一种非 0 的排列方式且其行列式的值 s 为 1 或 −1。同样因为 $\boldsymbol{U}$ 为上三角矩阵，所以其行列式的值为该矩阵主对角线元素之积，即 $\boldsymbol{A}$ 矩阵行列式为 $\det(\boldsymbol{A})=s\prod_{i=1}^n u_{ii}$。MATLAB 提供了内在函数 `det(A)`，利用它可以直接求取矩阵 `A` 的行列式。

〖例 3.1〗 假设给出如下的矩阵 $\boldsymbol{A}$

$$\boldsymbol{A}=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 0\end{bmatrix}$$

则由下面 MATLAB 语句可以容易地求出该矩阵的行列式为：

```
>> A = [1,2,3; 4 5 6; 7 8 0]; det(A)
ans =
     27
```

为统一起见，在本节的其他部分中将直接对同样的矩阵 $\boldsymbol{A}$ 求取其他矩阵参数。

- **矩阵的迹** (trace)

假设一个方阵为 $\boldsymbol{A}=\{a_{ij}\}$, $i,j=1,2,\cdots,n$，则矩阵 $\boldsymbol{A}$ 的迹定义为：

$$\mathrm{tr}(\boldsymbol{A})=\sum_{i=1}^n a_{ii} \tag{3.8}$$

亦即矩阵的迹为该矩阵对角线上各个元素之和。由代数理论可知，矩阵的迹和该矩阵的特征值之和是相同的，矩阵 `A` 的迹可以由 MATLAB 函数 `trace(A)` 求出。例 3.1 中矩阵的迹可以由下面的 MATLAB 语句直接求出

```
>> trace(A)
ans =
       6
```

- **矩阵的秩** (rank)

若矩阵所有的列向量中共有 r_c 个线性无关，则称矩阵的列秩为 r_c，如果 $r_c=m$，则称 $\boldsymbol{A}$ 为列满秩矩阵。相应地，若矩阵 $\boldsymbol{A}$ 的行向量中有 r_r 个是线性无关的，则称矩阵 $\boldsymbol{A}$ 的行秩为 r_r。如果 $r_r=n$，则称 $\boldsymbol{A}$ 为行满秩矩阵。可以证明，矩阵的行秩和列秩是相等的，故称之为矩阵的秩，记作：

$$\mathrm{rank}(\boldsymbol{A})=r_c=r_r \tag{3.9}$$

这时矩阵的秩为 rank($\boldsymbol{A}$)。矩阵的秩也表示该矩阵中行列式不等于 0 的子式的最大阶次，所谓子式，即为从原矩阵中任取 k 行及 k 列所构成的子矩阵。

矩阵求秩的算法也是多种多样的，其区别是有的算法是稳定的，而有的算法可能因矩阵的条件数变化不是很稳定。MATLAB 中采用的算法是基于矩阵的奇异值分解的算法[8]：首先对矩阵作奇异值分解，得出矩阵 $\boldsymbol{A}$ 的 n 个奇异值 $\sigma_i, i=1,2,\cdots,n$，在这 n 个奇异值中找出大于给定误差限 ε 的个数 r，这时 r 就可以认为是 $\boldsymbol{A}$ 矩阵的秩。

MATLAB 提供了一个内在函数 `rank(A,ε)` 来用数值方法求取一个已知矩阵 $\boldsymbol{A}$ 的数值秩，其中 ε 为机器精度。如果没用特殊说明，可以由 `rank(A)` 求出 `A` 矩阵的秩。例 3.1 中矩阵 $\boldsymbol{A}$ 的秩可以由下面的 MATLAB 语句直接求出。

```
>> rank(A)
ans =
       3
```

- **矩阵的范数** (norm)

矩阵的范数是对矩阵的一种测度，在介绍矩阵的范数之前，首先要介绍向量范数的基本概念。

如果对线性空间中的一个向量 $\boldsymbol{x}$ 存在一个函数 $\rho(\boldsymbol{x})$ 满足下面 3 个条件：

(1) $\rho(\boldsymbol{x}) \geq 0$ 且 $\rho(\boldsymbol{x})=0$ 的充要条件是 $\boldsymbol{x}=\boldsymbol{0}$

(2) $\rho(a\boldsymbol{x})=|a|\rho(\boldsymbol{x})$, a 为任意标量

(3) 对向量 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 有 $\rho(\boldsymbol{x}+\boldsymbol{y}) \leq \rho(\boldsymbol{x})+\rho(\boldsymbol{y})$

则称 $\rho(\boldsymbol{x})$ 为 $\boldsymbol{x}$ 向量的范数。范数的形式是多种多样的，可以证明，下面给出的一族式子都满足上述的 3 个条件

$$\|\boldsymbol{x}\|_p=\left(\sum_{i=1}^{n}|x_i|^p\right)^{1/p},\ p=1,2,\cdots,\text{且}\ \|\boldsymbol{x}\|_\infty=\max_{1\leq i\leq n}|x_i| \tag{3.10}$$

这里用到了向量范数的记号 $\|\boldsymbol{x}\|_p$。

矩阵的范数定义比向量的稍复杂一些，其完全的定义如下：对于任意的非零向量 $\boldsymbol{x}$，矩阵 $\boldsymbol{A}$ 的范数为

$$\|\boldsymbol{A}\|=\sup_{\boldsymbol{x}\neq 0}\frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \tag{3.11}$$

和向量的范数一样，对矩阵来说也有常用的范数定义方法

$$\|\boldsymbol{A}\|_1=\max_{1\leq j\leq n}\sum_{i=1}^{n}|a_{ij}|,\ \|\boldsymbol{A}\|_2=\sqrt{s_{\max}(\boldsymbol{A}^T\boldsymbol{A})},\ \|\boldsymbol{A}\|_\infty=\max_{1\leq i\leq n}\sum_{j=1}^{n}|a_{ij}| \tag{3.12}$$

其中 $s(\boldsymbol{X})$ 为 $\boldsymbol{X}$ 矩阵的特征值，而 $s_{\max}(\boldsymbol{A}^T\boldsymbol{A})$ 即为 $\boldsymbol{A}^T\boldsymbol{A}$ 矩阵的最大特征值。事实上，$\|\boldsymbol{A}\|_2$ 还等于 $\boldsymbol{A}$ 矩阵的最大奇异值。

MATLAB 提供了求取矩阵范数的函数 `norm()`，允许求各种意义下的矩阵范数。该函数的调用格式为 `N = norm(A, 选项)`，其中允许的选项如表 3-1 所示。这样例 3.1 中矩阵 $\boldsymbol{A}$ 的各种范数可以由下面的 MATLAB 函数直接求出

表 3-1 矩阵范数函数的选项表

选项	意义及算法
无	矩阵的最大奇异值，即 $\|\boldsymbol{A}\|_2$
2	与默认调用方式相同，亦为即 $\|\boldsymbol{A}\|_2$
1	矩阵的 1-范数，即 $\|\boldsymbol{A}\|_1$
Inf 或 'inf'	矩阵的无穷范数，即 $\|\boldsymbol{A}\|_\infty$
'fro'	矩阵的 F-范数，即 $\|\boldsymbol{A}\|_F = \sqrt{\sum(\boldsymbol{A}^T\boldsymbol{A})_{ii}}$
数值 P	对向量可取任何整数，而对矩阵只可取 1, 2, inf 或 'fro'
-inf	只可用于向量，$\|\boldsymbol{A}\|_{-\infty} = \min(\|\sum a_i\|)$

```
>> [norm(A), norm(A,2), norm(A,1), norm(A,Inf), norm(A,'fro')]
ans =
     13.2015    13.2015    15.0000    15.0000    14.2829
```

这里有两点值得注意：首先 norm(A) 和 norm(A,2) 应该给出同样的结果，因为它们都表示 $\|\boldsymbol{A}\|_2$；其次因为巧合，在这个例子中 $\|\boldsymbol{A}\|_1 = \|\boldsymbol{A}\|_\infty$ (对称矩阵也能得出这样的结论)。但一般情况下，$\|\boldsymbol{A}\|_1 \neq \|\boldsymbol{A}\|_\infty$。

- **矩阵的特征多项式** (characteristic polynomial) 、**特征方程与特征根** (eigenvalues)

构造一个矩阵 $s\boldsymbol{I} - \boldsymbol{A}$，并求出该矩阵的行列式，则可以得出一个多项式 $C(s)$

$$C(s) = \det(s\boldsymbol{I} - \boldsymbol{A}) = s^n + c_1 s^{n-1} + \cdots + c_{n-1}s + c_n \tag{3.13}$$

这样的多项式 $C(s)$ 称为矩阵 $\boldsymbol{A}$ 的特征多项式，其中系数 $c_i, i = 1, 2, \cdots, n$ 称为矩阵的特征多项式系数。

MATLAB 提供了求取矩阵特征多项式系数的函数 C=poly(A)，而返回的 C 为一个行向量，其各个分量为矩阵 A 的降幂排列的特征多项式系数。该函数的另外一种调用格式是：如果给定的 A 为向量，则假定该向量是一个矩阵的特征根，由此求出该矩阵的特征多项式系数；如果向量 A 中有无穷大或 NaN 值，则首先剔除它。

〖例 3.2〗 考虑例 3.1 中给出的矩阵 A，直接调用 MATLAB 函数 poly(A) 则可以求出该矩阵的特征多项式系数向量为

```
>> B = poly(A)
B =
    1.0000   -6.0000  -72.0000  -27.0000
```

由前面得出的结果可见，此矩阵的特征多项式可以写成 $P(s) = s^3 - 60s^2 - 72s - 27$。事实上，$P(s)$ 多项式即为原矩阵特征多项式的理论解。由 poly() 函数调用结果产生的相对误差为

```
>> P = [1, -6 -72, -27]; norm((P-B)./P)
ans =
```

```
    5.6538e-015
```

由上面的结果可见，MATLAB 提供的 poly() 函数在计算矩阵的特征多项式系数时会产生微小的误差。阅读 poly.m 文件可以发现该函数利用了 eig() 函数来求特征值，而 eig() 函数的计算是很复杂的迭代过程，产生些小误差在所难免，而这一误差最终又被传递到其他函数的计算中去，所以在使用时应该注意。

在实际应用中还有其他简单的方法可以求出矩阵的特征多项式系数，如下面给出的 Fadeev-Fadeeva 递推算法也可以求出矩阵的特征多项式。

$$\begin{cases} c_k = -\dfrac{1}{k}\mathrm{tr}(\boldsymbol{A}\boldsymbol{R}_k), & k=1,2,\cdots,n \\ \boldsymbol{R}_1=\boldsymbol{I},\ \boldsymbol{R}_k=\boldsymbol{A}\boldsymbol{R}_{k-1}+c_{k-1}\boldsymbol{I}, & k=2,\cdots,n \end{cases} \tag{3.14}$$

该算法首先给出一个单位阵 $\boldsymbol{I}$，并将之赋给 $\boldsymbol{R}_1$，然后对每个 k 的值分别求出特征多项式参数，并更新 $\boldsymbol{R}_k$ 矩阵，最终得出矩阵的特征多项式系数 c_k。该算法可以直接由下面的 MATLAB 语句编写一个 poly1() 函数实现。

```
function c=poly1(A)
[nr,nc]=size(A);
if nc==nr % 给出若为方阵，则用 Fadeev-Fadeeva 算法求特征多项式
   I=eye(nc);  R=I; c=[1 zeros(1,nc)];
   for k=1:nc
      c(k+1)=-1/k*trace(A*R);
      R=A*R+c(k+1)*I;
   end
elseif (nr==1 | nc==1) % 给出为向量时，构造矩阵
   A=A(isfinite(A)); n=length(A); % 除去非数或无界的特征根
   c = [1 zeros(1,n)];
   for j=1:n
      c(2:(j+1))=c(2:(j+1))-A(j).*c(1:j);
   end
else   % 参数有误则给出错误信息
   error('Argument must be a vector or a square matrix.')
end
```

其中后面的语句对应于函数的另外一种调用方法。同样考虑前面的例子，如果调用上面的程序段，则可以得出如下的结果

```
>> c=poly1(A)
c =
     1    -6   -72   -27
```

可见这样得出的特征多项式的系数均为整数 (精确解)，此外由于这里给出的算法运算起来比

较简单，不需要求取矩阵的特征值，所以运算量也比 poly() 的小，且可以得出精确的解。

令特征多项式等于零所构成的方程称为该矩阵的特征方程，而特征方程的根称为该矩阵的特征根。特征根当然可以由后面将要介绍的矩阵特征值算法直接求出，如果获得了矩阵的特征方程，则矩阵的特征根还可以通过求解多项式方程而求出。这可以调用 MATLAB 函数 V = roots(P) 而直接获得，其中，V 为特征方程式的解，即原矩阵的特征根。例 3.1 中矩阵的特征根可以由下面的 MATLAB 语句直接求出：

```
>> roots(c)
ans =
   12.1229
   -5.7345
   -0.3884
```

- **多项式及多项式矩阵的求值**

多项式的求值可以由polyval()函数直接完成，对于多项式矩阵来说，则可以由 polyvalm() 函数来完成，这两个函数的调用格式为 C=polyval(aa,x) 或 B=polyvalm (aa,A)，其中 aa 为多项式系数降幂排列构成的向量，即 aa=$[a_1,a_2,\cdots,a_n,a_{n+1}]$，x 为一个标量，而 A 为一个给定矩阵，这时返回的矩阵 B 为下面的矩阵多项式的值

$$\boldsymbol{B}=a_1\boldsymbol{A}^n+a_2\boldsymbol{A}^{n-1}+\cdots+a_n\boldsymbol{A}+a_{n+1}\boldsymbol{I} \tag{3.15}$$

其中 $\boldsymbol{I}$ 为和 $\boldsymbol{A}$ 同阶次的单位矩阵，而 C=a_1 x.^n + $\cdots$ + a_{n+1}。

【例 3.3】 Hamilton-Cailey 定理是矩阵理论中的一个比较重要的定理，它的内容为：若矩阵 $\boldsymbol{A}$ 的特征多项式为

$$\det(s\boldsymbol{I}-\boldsymbol{A})=a_1s^n+a_2s^{n-1}+\cdots+a_ns+a_{n+1} \tag{3.16}$$

则有

$$a_1\boldsymbol{A}^n+a_2\boldsymbol{A}^{n-1}+\cdots+a_n\boldsymbol{A}+a_{n+1}\boldsymbol{I}=\boldsymbol{0} \tag{3.17}$$

假设矩阵 $\boldsymbol{A}$ 由例 3.1 给出，则可以由下面的 MATLAB 语句来验证 Hamilton-Cailey 定理：

```
>> A=[1,2,3; 4,5,6; 7,8,0];
   aa=poly(A); B=polyvalm(aa, A);  norm(B)
ans =
    2.9932e-013
```

由于使用的 poly() 函数会产生一定的误差，所以得出的 B 矩阵并不是很精确。如果将上面语句中 poly() 函数用我们编写的 poly1() 代替，则

```
>> aa1=poly1(A); B1=polyvalm(aa1, A);  norm(B1)
ans =
     0
```

则由此得出的 B 矩阵就会完全等于 0，这样就由该矩阵验证了 Hamilton-Cailey 定理。

3.2.3 矩阵的相似变换与分解

3.2.3.1 矩阵的相似变换与正交变换

假设有一个 $n\times n$ 的方阵 $\boldsymbol{A}$，并存在一个和它同阶的非奇异矩阵 $\boldsymbol{T}$，则可以对 $\boldsymbol{A}$ 矩阵进行如下的变换

$$\widehat{\boldsymbol{A}}=\boldsymbol{T}^{-1}\boldsymbol{A}\boldsymbol{T} \tag{3.18}$$

这种变换称为 $\boldsymbol{A}$ 的相似变换 (similarity transform)。可以证明，变换后的矩阵 $\widehat{\boldsymbol{A}}$ 的特征值和原矩阵 $\boldsymbol{A}$ 是一致的，亦即相似变换并不改变原矩阵的特征结构。

对于一类特殊的相似变换矩阵 $\boldsymbol{T}$ 来说，如果它本身满足 $\boldsymbol{T}^{-1}=\boldsymbol{T}^*$，其中 $\boldsymbol{T}^*$ 为 $\boldsymbol{T}$ 的 Hermit 共轭转置矩阵，则称 $\boldsymbol{T}$ 为正交矩阵，并将之记为 $\boldsymbol{Q}=\boldsymbol{T}$。可见正交矩阵 $\boldsymbol{Q}$ 满足下面的条件

$$\boldsymbol{Q}^*\boldsymbol{Q}=\boldsymbol{I},\ 且\ \boldsymbol{Q}\boldsymbol{Q}^*=\boldsymbol{I} \tag{3.19}$$

其中 $\boldsymbol{I}$ 为 $n\times n$ 的单位阵。

正交矩阵中还有一类特殊的形式，如果 $\boldsymbol{A}$ 矩阵不是满秩矩阵，且 $\boldsymbol{Z}$ 矩阵为正交矩阵，亦即它满足 $\boldsymbol{Z}^*\boldsymbol{Z}=\boldsymbol{I}$，如果矩阵 $\boldsymbol{Z}$ 可以使得 $\boldsymbol{A}\boldsymbol{Z}=\boldsymbol{0}$，则称 $\boldsymbol{Z}$ 矩阵为化零空间 (null space)。

MATLAB 中提供了求取正交矩阵和化零矩阵的函数 `orth()` 和 `null()`，这两个矩阵的调用方式分别为 `Q = orth(A)` 及 `Z = null(A)`。其中前一个函数由矩阵 $\boldsymbol{A}$ 构成一个正交基，亦即它的各个列可以张成 $\boldsymbol{A}$ 矩阵的各列同样的空间，且 $\boldsymbol{Q}$ 的各列为正交的。调用 MATLAB 提供的 `null()` 函数可以获得前面提及的化零空间，如果 $\boldsymbol{A}$ 为满秩矩阵，则不存在这样的矩阵 $\boldsymbol{Z}$，这时 `null()` 函数将返回一个空的矩阵。

【例 3.4】 重新考虑例 3.1 中给出的矩阵 $\boldsymbol{A}$，可以通过下面的 MATLAB 语句求取并检验其正交基矩阵

```
>> Q=orth(A)
Q =
    0.2304    0.3961   -0.8889
    0.6073    0.6552    0.4493
    0.7604   -0.6433   -0.0896
>> I = eye(size(A));  norm(Q*Q'-I)
ans =
     5.6023e-016
>> norm(Q'*Q-I)
ans =
     5.1660e-016
```

可见，通过这样的函数可以建立一个正交矩阵 $\boldsymbol{Q}$，且这一矩阵满足式 (3.19) 中规定的条件。

3.2.3.2 矩阵的三角分解及 Cholesky 分解

矩阵的三角分解又称为 LU 分解，它的目的是将一个矩阵分解成一个下三角矩阵 $\boldsymbol{L}$ 和一个上三角矩阵 $\boldsymbol{U}$ 的乘积，亦即 $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{U}$，其中 $\boldsymbol{L}$ 和 $\boldsymbol{U}$ 矩阵可以分别写成

$$\boldsymbol{L}=\begin{bmatrix}1 & & & \\ l_{21} & 1 & & \\ \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & \cdots & 1\end{bmatrix},\quad \boldsymbol{U}=\begin{bmatrix}u_{11} & u_{12} & \cdots & u_{1n} \\ & u_{22} & \cdots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn}\end{bmatrix} \tag{3.20}$$

由这两个矩阵可以简单地写出一个矩阵 $\boldsymbol{A}^F$，其中

$$\boldsymbol{A}^F=\boldsymbol{L}+\boldsymbol{U}-\boldsymbol{I}=\begin{bmatrix}u_{11} & u_{12} & \cdots & u_{1n} \\ l_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & u_{nn}\end{bmatrix} \tag{3.21}$$

这样产生的矩阵与原来的 $\boldsymbol{A}$ 矩阵的关系可以写成

$$\begin{array}{llll} a_{11}=u_{11}, & a_{12}=u_{12}, & \cdots & a_{1n}=u_{1n} \\ a_{21}=l_{21}u_{11}, & a_{22}=l_{21}u_{12}+u_{22}, & \cdots & u_{2n}=l_{21}u_{1n}+u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}=l_{n1}u_{11}, & a_{n2}=l_{n1}u_{12}+l_{n2}u_{22}, & \cdots & a_{nn}=\sum\limits_{k=1}^{n-1}l_{nk}u_{kn}+u_{nn} \end{array} \tag{3.22}$$

由上式可以立即得出求取 l_{ij} 和 u_{ij} 的递推计算公式

$$l_{ij}=\frac{a_{ij}-\sum\limits_{k=1}^{j-1}l_{ik}u_{kj}}{u_{jj}},\ (j<i),\ 及\ u_{ij}=a_{ij}-\sum_{k=1}^{i-1}l_{ik}u_{kj},\ (j\geqslant i) \tag{3.23}$$

该公式的递推初值为

$$u_{1i}=a_{1i},\ i=1,2,\cdots,n \tag{3.24}$$

注意，在上述的算法中并未对主元素进行任何选取，因此该算法并不一定数值稳定。在 MATLAB 下也给出了矩阵的 LU 分解函数 `lu()`，该函数的调用格式为 `[L, U] = lu(A)`，其中 `L`, `U` 分别为变换后的下三角和上三角矩阵。在 MATLAB 的 `lu()` 函数中考虑了主元素选取的问题，所以该函数一般会给出可靠的结果。由该函数得出的下三角矩阵 $\boldsymbol{L}$ 并不一定是一个真正的下三角矩阵，因为选取它可能进行了一些元素行的交换，这样主对角线的元素可能不是 1，而在矩阵 $\boldsymbol{L}$ 内存在一个惟一的如式 (3.7) 中定义的置换，其各个元素的值均是 1。如果想获得有关换行信息，则可以由下面的格式调用该函数 `[L, U, P] = lu(A)`，式中 `P` 为排列规则矩阵，而这时 `L` 和 `U` 分别为真正的下三角和上三角矩阵。但使用这一调用规则时一定要注意，若 `P` 不为单位阵时，得出的 $\boldsymbol{L}$ 和 $\boldsymbol{U}$ 矩阵不满足 $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{U}$，而满足 $\boldsymbol{A}=\boldsymbol{P}^{-1}\boldsymbol{L}\boldsymbol{U}$。

〖例 3.5〗 再考虑例 3.1 中矩阵的 LU 分解问题。分别用两种方法调用 MATLAB 下的 lu() 函数，则可以得出的不同结果。

```
>> A=[1, 2, 3; 4, 5, 6; 7, 8, 0]; [L1,U1]=lu(A)
L1 =
    0.1429    1.0000         0
    0.5714    0.5000    1.0000
    1.0000         0         0
U1 =
    7.0000    8.0000         0
         0    0.8571    3.0000
         0         0    4.5000
>> L1*U1
ans =
     1     2     3
     4     5     6
     7     8     0
>> [L, U, P] = lu(A)
L =
    1.0000         0         0
    0.1429    1.0000         0
    0.5714    0.5000    1.0000
U =
    7.0000    8.0000         0
         0    0.8571    3.0000
         0         0    4.5000
P =
     0     0     1
     1     0     0
     0     1     0
>> L*U  % 在这样的分解下其乘积不等于A
ans =
     7     8     0
     1     2     3
     4     5     6
>> inv(P)*L*U   % 考虑到P矩阵后其总乘积等于A
ans =
     1     2     3
     4     5     6
```

```
     7     8     0
```

注意，这里得出的 $\boldsymbol{P}$ 矩阵不是一个单位矩阵，所以在进行计算时由于考虑主元素的原因对原来的排列进行了改动，这样，前一种方法得出的 $\boldsymbol{L}$ 也不是一个真正的下三角矩阵。在后一种调用方式中，注意 $\boldsymbol{LU} \neq \boldsymbol{A}$。

如果 $\boldsymbol{A}$ 矩阵为对称矩阵，则仍然可以用 LU 分解的方法对之进行分解，对称矩阵 LU 分解有特殊的性质，即 $\boldsymbol{L} = \boldsymbol{U}^T$，令 $\boldsymbol{D} = \boldsymbol{L}$ 为一个下三角矩阵，则可以将原来矩阵 $\boldsymbol{A}$ 分解成

$$\boldsymbol{A} = \boldsymbol{D}\boldsymbol{D}^T = \begin{bmatrix} d_{11} & & & \\ d_{21} & d_{22} & & \\ \vdots & \vdots & \ddots & \\ d_{n1} & d_{n2} & \cdots & d_{nn} \end{bmatrix} \begin{bmatrix} d_{11} & d_{21} & \cdots & d_{n1} \\ & d_{22} & \cdots & d_{n2} \\ & & \ddots & \vdots \\ & & & d_{nn} \end{bmatrix} \tag{3.25}$$

其中 $\boldsymbol{D}$ 矩阵可以形象地理解为原 $\boldsymbol{A}$ 矩阵的平方根。对该对称矩阵进行分解可以采用 Cholesky 分解算法，其具体叙述如下

$$d_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} d_{ik}^2},\ \ d_{ij} = \frac{a_{ij} - \sum\limits_{k=1}^{j-1} d_{ik}d_{jk}}{l_{jj}},\ (j < i) \tag{3.26}$$

MATLAB 提供了 `chol()` 函数来求取矩阵的 Cholesky 分解矩阵 $\boldsymbol{D}$，该函数的调用格式可以写成 `[D, P] = chol(A)`，式中返回的 D 为 Cholesky 分解矩阵，且 $\boldsymbol{A} = \boldsymbol{D}\boldsymbol{D}^T$；而 P-1 为 $\boldsymbol{A}$ 矩阵中正定的子矩阵的阶次，如果 $\boldsymbol{A}$ 为正定矩阵，则返回 $\boldsymbol{P} = 0$。当然也可以直接由 `D=chol(A)` 来调用该函数，这时要求 $\boldsymbol{A}$ 为一个正定矩阵。如果 $\boldsymbol{A}$ 不是正定矩阵，则这样调用将给出一个错误信息。

【例 3.6】 考虑一个对称的 3 阶 Hilbert 矩阵 $\boldsymbol{A}$，调用 MATLAB 的 `chol()` 函数可以得出如下结果

```
>> A=hilb(3); [D, P]=chol(A)
D =
    1.0000    0.5000    0.3333
         0    0.2887    0.2887
         0         0    0.0745
P =
     0
```

可见式中 $\boldsymbol{P} = 0$，这表示 $\boldsymbol{A}$ 阵是一个正定矩阵。如果试图对一个非正定矩阵进行 Cholesky 分解，则将得出错误信息，所以，`chol()` 函数还可以用来判定矩阵是否为正定矩阵。

3.2.3.3 矩阵的奇异值分解

矩阵的奇异值也可以看成是矩阵的一种测度。对任意的 $n \times m$ 矩阵 $\boldsymbol{A}$ 来说，总有

$$\boldsymbol{A}^T\boldsymbol{A} \geq \boldsymbol{0},\ \ \boldsymbol{A}\boldsymbol{A}^T \geq \boldsymbol{0} \tag{3.27}$$

且有

$$\operatorname{rank}(\boldsymbol{A}^T\boldsymbol{A}) = \operatorname{rank}(\boldsymbol{A}\boldsymbol{A}^T) = \operatorname{rank}(\boldsymbol{A}) \tag{3.28}$$

进一步可以证明，$\boldsymbol{A}^T\boldsymbol{A}$ 与 $\boldsymbol{A}\boldsymbol{A}^T$ 有相同的非负特征值 λ_i，在数学上把这些非负的特征值的平方根称作矩阵 $\boldsymbol{A}$ 的奇异值，记 $\sigma_i(A) = \sqrt{\lambda_i(\boldsymbol{A}^T\boldsymbol{A})}$。

假设 $\boldsymbol{A}$ 矩阵为 $n \times m$ 矩阵，且 $\operatorname{rank}(\boldsymbol{A}) = r$，则 $\boldsymbol{A}$ 矩阵可以分解为

$$\boldsymbol{A} = \boldsymbol{L}\begin{bmatrix} \boldsymbol{\Delta} & 0 \\ 0 & 0 \end{bmatrix}\boldsymbol{M}^T \tag{3.29}$$

其中 $\boldsymbol{L}$ 和 $\boldsymbol{M}$ 为正交矩阵，$\boldsymbol{\Delta} = \operatorname{diag}(\sigma_1, \cdots, \sigma_r)$ 为对角矩阵，其对角元素 $\sigma_1, \sigma_2, \cdots, \sigma_r$ 满足不等式 $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$。

MATLAB 提供了直接求取矩阵奇异值分解的函数，其调用方式为 `[L, A1, M] = svd(A)`。其中，`A` 为原始矩阵，返回的 `A1` 为对角矩阵，而 `L` 和 `M` 均为正交变换矩阵，并满足 $\boldsymbol{A} = \boldsymbol{L}\,\boldsymbol{A}_1\,\boldsymbol{M}^T$。

矩阵的奇异值大小通常决定矩阵的性态，如果矩阵的奇异值的差异特别大，则矩阵中某个元素有一个微小的变化将严重影响到原矩阵的参数，这样的矩阵又称为病态矩阵或坏条件矩阵，而在矩阵存在等于 0 的奇异值时称为奇异矩阵。矩阵最大奇异值 $\sigma_{\max}$ 和最小奇异值 $\sigma_{\min}$ 的比值又称为该矩阵的条件数，记作 $\operatorname{cond}(\boldsymbol{A})$，即 $\operatorname{cond}(\boldsymbol{A}) = \sigma_{\max}/\sigma_{\min}$，矩阵的条件数越大，则对元素变化越敏感。矩阵的最大和最小奇异值还分别经常记作 $\bar{\sigma}(\boldsymbol{A})$ 和 $\underline{\sigma}(\boldsymbol{A})$。在 MATLAB 下也提供了函数 `cond(A)` 来求取矩阵 $\boldsymbol{A}$ 的条件数。

【例 3.7】 *考虑例 3.1 中给出的 $\boldsymbol{A}$ 矩阵，如果调用 MATLAB 中给出的矩阵奇异值分解函数 `svd()`，则可以容易地求出 $\boldsymbol{L}, \boldsymbol{A}_1$ 和 $\boldsymbol{M}$ 矩阵，并可以容易地求出该矩阵的条件数。*

```
>> [L, A1, M]=svd(A)
L =
    0.2304    0.3961   -0.8889
    0.6073    0.6552    0.4493
    0.7604   -0.6433   -0.0896
A1 =
   13.2015         0         0
         0    5.4388         0
         0         0    0.3760
M =
    0.6046   -0.2733    0.7482
    0.7257   -0.1982   -0.6589
    0.3284    0.9413    0.0784
>> B = A'*A; C=sqrt(eig(B)); [cond(A), A1(1)/A1(end) C(end)/C(1)]
ans =
   35.1059   35.1059   35.1059
```

这里用到了三种方式求取矩阵的条件数，得出的结果是完全一致的。

〖例 3.8〗 对于 $n \neq m$ 的矩阵 $\boldsymbol{A}$ 来说，也可以对之作奇异值分解，例如

$$\boldsymbol{A}=\begin{bmatrix}1 & 3 & 5 & 7\\2 & 4 & 6 & 8\end{bmatrix}$$

使用如下命令则可以得出

```
>> A = [1, 3, 5, 7; 2, 4, 6, 8]; [L, A1, M] = svd(A)
L =
    0.6414    0.7672
    0.7672   -0.6414
A1 =
   14.2691         0         0         0
         0    0.6268         0         0
M =
    0.1525   -0.8226    0.5477         0
    0.3499   -0.4214   -0.7303    0.4082
    0.5474   -0.0201   -0.1826   -0.8165
    0.7448    0.3812    0.3651    0.4082
>> L*A1*M'
ans =
    1.0000    3.0000    5.0000    7.0000
    2.0000    4.0000    6.0000    8.0000
>> norm(A-ans)
ans =
    2.6295e-015
```

对这个例子进行逆运算，即 $\boldsymbol{L}\boldsymbol{A}_1\boldsymbol{M}^T$，则可以还原成原来的 $\boldsymbol{A}$ 矩阵，由前面的分析可见，这样得出的矩阵误差是很小的。

3.2.4 矩阵的特征值与特征向量

对一个矩阵 $\boldsymbol{A}$ 来说，如果存在一个非零的向量 $\boldsymbol{x}$，且有一个标量 λ 满足

$$\boldsymbol{A}\boldsymbol{x}=\lambda\boldsymbol{x} \tag{3.30}$$

则称 λ 为 $\boldsymbol{A}$ 矩阵的一个特征值，而 $\boldsymbol{x}$ 为对应于特征值 λ 的特征向量，严格说来，$\boldsymbol{x}$ 应该称为 $\boldsymbol{A}$ 的右特征向量。如果矩阵 $\boldsymbol{A}$ 的特征值不包含重复的值，则对应的各个特征向量为线性无关的，这样由各个特征向量可以构成一个非奇异的矩阵，如果用它对原始矩阵作相似变换，则可以得出一个对角矩阵。矩阵的特征值与特征向量由 MATLAB 提供的函数 eig() 可以容易地求出，该函数的调用格式为 [V, D] = eig(A)，其中 A 为要处理的矩阵，D 为一个对角矩阵，其对角线上的元素为矩阵 $\boldsymbol{A}$ 的特征值，而每个特征值对应的 V

矩阵的列为该特征值的特征向量，该矩阵是一个满秩矩阵。MATLAB 的矩阵特征值的结果满足 $\boldsymbol{AV}=\boldsymbol{VD}$，且每个特征向量各元素的平方和 (即 2 范数) 均为 1。如果调用该函数时只给出一个返回变量，则将只返回矩阵 $\boldsymbol{A}$ 的特征值。即使 $\boldsymbol{A}$ 为复数矩阵，也照样可以由 eig() 函数得出其特征值与特征向量矩阵的。

MATLAB 中的 eig() 函数是基于两步 QR 算法实现的。在矩阵含有重特征根时，特征向量矩阵可能趋于奇异，所以在使用此函数时应该注意。

矩阵特征值的求解算法是多种多样的，最常用的有求解实对称矩阵特征值与特征向量的 Jacobi 算法，有原点平移 QR 分解法与两步 QR 算法，矩阵的特征值与特征向量的求解有许多标准的子程序或程序库可以直接调用，如著名的 EISPACK 软件包[13, 44] 等。

〖例 3.9〗 考虑例 3.1 中给出的矩阵 $\boldsymbol{A}$，如果调用 eig() 函数则可以获得矩阵 $\boldsymbol{A}$ 的特征值与特征向量矩阵。

```
>> A=[1, 2, 3; 4, 5, 6; 7, 8, 0]; [v, d] = eig(A)
v =
    0.7471   -0.2998   -0.2763
   -0.6582   -0.7075   -0.3884
    0.0931   -0.6400    0.8791
d =
   -0.3884         0         0
         0   12.1229         0
         0         0   -5.7345
>> max(norm(A*v-v*d))
ans =
     5.3986e-015
>> eig(A)
ans =
   -0.3884
   12.1229
   -5.7345
```

可见在前面的例子中两次调用了 eig() 函数，但由于返回参数个数不一致，所以前面的调用返回矩阵 $\boldsymbol{A}$ 的特征值与特征向量，而后面的调用只返回了矩阵 $\boldsymbol{A}$ 的特征值而不返回特征向量矩阵。另外，返回特征值的格式也因返回变量个数不同而不同。

若想由 C 或 Fortran 语句从最底层编程，则需要编写相当大的程序才可以完成特征值、特征向量的计算，例如 EISPACK[44] 中提供的子程序源程序有 500 多条。

MATLAB 还提供了求取广义特征值的方法。广义特征值的概念如下：假设存在一个标量 λ 和一个非零向量 $\boldsymbol{x}$，使得

$$\boldsymbol{Ax}=\lambda\boldsymbol{Bx} \tag{3.31}$$

成立，则 λ 称为广义特征值，而 $\boldsymbol{x}$ 向量称为广义特征向量。事实上，普通的矩阵特征值

问题可以看成是广义特征值问题的一个特例，因为若假定 $\boldsymbol{B}=\boldsymbol{I}$ 为单位阵，则式 (3.31) 中的形式可以直接转化成普通矩阵特征值问题。

如果 $\boldsymbol{B}$ 矩阵为一个非奇异方阵，则上面的方程可以容易地转换成一般矩阵 $\boldsymbol{B}^{-1}\boldsymbol{A}$ 的特征值问题

$$\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{x}=\lambda\boldsymbol{x} \tag{3.32}$$

即 λ 和 $\boldsymbol{x}$ 分别为 $\boldsymbol{B}^{-1}\boldsymbol{A}$ 矩阵的特征值和特征向量。但一般情况下不能随便假设 $\boldsymbol{B}$ 阵为非奇异的方阵，所以文献 [35] 中给出了广义特征值问题的 QZ 算法。在 MATLAB 中给出的 eig() 函数可以直接用来求取矩阵的广义特征值和特征向量，这时的调用格式为 [V, D] = eig(A, B)，这一函数仍返回一个特征向量矩阵 $\boldsymbol{V}$ 及一个对角型特征值矩阵 $\boldsymbol{D}$，满足 $\boldsymbol{AV}=\boldsymbol{BVD}$。值得指出的是，eig() 函数也能较好地解决 $\boldsymbol{B}$ 为奇异矩阵时的广义特征值问题。

【例 3.10】 假设给出如下的矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$

$$\boldsymbol{A}=\begin{bmatrix}5&7&6&5\\7&10&8&7\\6&8&10&9\\5&7&9&10\end{bmatrix},\quad B=\begin{bmatrix}2&6&-1&-2\\5&-1&2&3\\-3&-4&1&10\\5&-2&-3&8\end{bmatrix}$$

则使用下列命令可以求出矩阵的广义特征值和特征向量。

```
>> A=[5,7,6,5; 7,10,8,7; 6,8,10,9; 5,7,9,10];
   B=[2,6,-1,-2; 5,-1,2,3; -3,-4,1,10; 5,-2,-3,8];
   [V, D] = eig(A, B)
V =
   0.2224 - 0.0000i  0.5157 - 0.0188i -0.0188 + 0.5157i  0.8328 + 0.0000i
   0.5985 - 0.0000i -0.1244 + 0.1375i  0.1375 - 0.1244i -0.5072 + 0.0000i
   0.4800 - 0.0000i -0.3400 - 0.5588i -0.5588 - 0.3400i -0.1929 - 0.0000i
   0.6016 - 0.0000i  0.0603 + 0.5175i  0.5175 + 0.0603i  0.1098 + 0.0000i
D =
   4.7564 + 0.0000i        0                  0                 0
        0             0.0471 + 0.1750i        0                 0
        0                  0             0.0471 - 0.1750i       0
        0                  0                  0             -0.0037
>> norm(A*V-B*V*D)
ans =
     1.5783e-014
```

3.2.5　矩阵求逆与线性方程求解

3.2.5.1　矩阵求逆运算与线性方程求解

对一个已知的 $n \times n$ 非奇异方阵 $\boldsymbol{A}$ 来说，如果有一个同样大小的 $\boldsymbol{C}$ 矩阵满足

$$\boldsymbol{AC} = \boldsymbol{CA} = \boldsymbol{I} \tag{3.33}$$

式中 $\boldsymbol{I}$ 为单位阵，则称 $\boldsymbol{C}$ 矩阵为 $\boldsymbol{A}$ 矩阵的逆矩阵，并记作 $\boldsymbol{C} = \boldsymbol{A}^{-1}$。

矩阵求逆运算往往和线性代数方程的求解有关，考虑下面给出的线性代数方程

$$\boldsymbol{Ax} = \boldsymbol{B} \tag{3.34}$$

式中 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 为相容维数的矩阵；$\boldsymbol{x}$ 亦为矩阵，它称为方程的解。如果 $\boldsymbol{A}$ 为非奇异的方阵，则可以立即得出方程的解为

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{B} \tag{3.35}$$

线性方程的求解和矩阵求逆的算法是多种多样的，比较常用的有全 (列) 主元素 Gauss 消去法、LU 分解法、基于奇异值分解的方法等。MATLAB 提供了一个求取逆矩阵的函数 `inv()`，其调用格式为 `B=inv(A)`。通过这一函数的调用就可以直接由给出的 A 矩阵求出其逆矩阵 B 来。在 MATLAB 下，式 (3.34) 中的解可以由 `x=inv(A)*B` 求出，也可以简单地由 `x=A\B` 求出。但是 `inv()` 函数的调用也有值得注意之处，例如若 A 矩阵为奇异的或接近奇异的，则利用此函数有可能产生错误的结果。

【例 3.11】 *再考虑例 3.1 中给出的矩阵，如果调用 MATLAB 的矩阵求逆函数 `inv()`，则可以立即得出该矩阵的逆矩阵来。*

```
>> A=[1 2 3; 4 5 6; 7 8 0]; inv(A)
ans =
   -1.7778    0.8889   -0.1111
    1.5556   -0.7778    0.2222
   -0.1111    0.2222   -0.1111
```

【例 3.12】 *如果原始矩阵 $\boldsymbol{A}$ 为下面的奇异矩阵*

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

则调用 MATLAB 的矩阵求逆函数可以得出下面的结果

```
>> A=[1, 2, 3; 4, 5, 6; 7, 8, 9]; B = inv(A)
Warning: Matrix is close to singular or badly scaled.
      Results may be inaccurate. RCOND = 2.055969e-018.
B =
   1.0e+016 *
    -0.4504    0.9007   -0.4504
```

```
    0.9007   -1.8014    0.9007
   -0.4504    0.9007   -0.4504
```

可见这时给出一个警告信息，提示用户该矩阵接近于奇异 (事实上 $\boldsymbol{A}$ 矩阵就是一个奇异矩阵，但通过数值算法后得出的是接近奇异的结论)，并提示得出的逆矩阵可能是不正确的，最后还列出了逆矩阵的结果 —— 该结果显然是没有意义的。

3.2.5.2 矩阵的广义逆

如果用户确实需要得出原来奇异矩阵的一种"逆"阵，那么就需要使用广义逆的概念了。对要研究的矩阵 $\boldsymbol{A}$，如果存在一个矩阵 $\boldsymbol{N}$，它满足

$$\boldsymbol{ANA}=\boldsymbol{A} \tag{3.36}$$

则 $\boldsymbol{N}$ 矩阵称为 $\boldsymbol{A}$ 的广义逆矩阵，记作 $\boldsymbol{N}=\boldsymbol{A}^{-}$。如果 $\boldsymbol{A}$ 阵是一个 $n\times m$ 的长方形矩阵。定义下面的范数最小化指标

$$\min_{\boldsymbol{x}}\|\boldsymbol{Ax}-\boldsymbol{B}\| \tag{3.37}$$

则满足这一指标的解共有无穷多个，可以证明，对一个给定的矩阵 $\boldsymbol{A}$，存在一个惟一的矩阵 $\boldsymbol{M}$ 使得下面 3 个条件同时成立：

(1) $\boldsymbol{AMA}=\boldsymbol{A}$

(2) $\boldsymbol{MAM}=\boldsymbol{M}$

(3) $\boldsymbol{AM}$ 与 $\boldsymbol{MA}$ 均为对称矩阵

这样的矩阵 $\boldsymbol{M}$ 称为矩阵 $\boldsymbol{A}$ 的 Moore-Penrose 广义逆矩阵，记作 $\boldsymbol{M}=\boldsymbol{A}^{+}$。从上面的 3 个条件中可以看出，第一个条件和一般广义逆的定义也是一样的，所不同的是它还要求满足第二和第三个条件，这样就会得出惟一的广义逆矩阵了。更进一步对复数矩阵 $\boldsymbol{A}$ 来说，若得出的广义逆矩阵的第二个条件扩展为 $\boldsymbol{MA}$ 与 $\boldsymbol{AM}$ 均为 Hermit 矩阵，则这样构造的矩阵也是惟一的。

MATLAB 提供了求取矩阵 Moore-Penrose 广义逆的函数 `pinv()`，该函数的调用格式为 `B = pinv(A, tol)`，其中 `tol` 为判 0 用误差限，如果省略此参数，则判 0 用误差限选用机器的精度 `eps`，这时将返回 `A` 的 Moore-Penrose 广义逆矩阵 `B`。如果 `A` 矩阵为非奇异方阵，则该函数得出的就是矩阵的逆阵。

〖例 3.13〗 考虑一个给定的长方形矩阵 $\boldsymbol{A}$

$$\boldsymbol{A}=\begin{bmatrix}6 & 1 & 4 & 2 & 1\\ 3 & 0 & 1 & 4 & 2\\ -3 & -2 & -5 & 8 & 4\end{bmatrix}$$

可以通过下面的 MATLAB 命令求出矩阵的秩、Moore-Penrose 广义逆，并分析得出的广义逆矩阵性质。

```
>> A=[6,1,4,2,1; 3,0,1,4,2; -3,-2,-5,8,4]; rank(A)
ans =
```

```
        2
>> iA = pinv(A) % 非满秩矩阵的广义逆
iA =
    0.0730    0.0413   -0.0221
    0.0108    0.0020   -0.0156
    0.0459    0.0178   -0.0385
    0.0327    0.0431    0.0638
    0.0164    0.0215    0.0319
>> B=iA*A*iA
ans =
    0.0730    0.0413   -0.0221
    0.0108    0.0020   -0.0156
    0.0459    0.0178   -0.0385
    0.0327    0.0431    0.0638
    0.0164    0.0215    0.0319
>> norm(iA-B)   % 测试关系式 iA*A*iA=iA
ans =
    6.6087e-017
>> norm(A*iA*A-A) % 测试关系式 A*iA*A=A
ans =
    5.8280e-015
>> norm(iA*A-A'*iA')  % 测试 iA*A 的对称性
ans =
    3.4454e-016
>> norm(A*iA-iA'*A')  % 测试 A*iA 的对称性
ans =
    3.8158e-016
```

可见由 MATLAB 直接得出的广义逆矩阵确实为 Moore-Penrose 逆。下面考虑对 A^+ 再求一次广义逆，并观察其结果。

```
>> iiA = pinv(iA)  % 对广义逆结果再进行广义逆
iiA =
    6.0000    1.0000    4.0000    2.0000    1.0000
    3.0000    0.0000    1.0000    4.0000    2.0000
   -3.0000   -2.0000   -5.0000    8.0000    4.0000
>> norm(iiA-A)     % 和原矩阵进行比较
ans =
    1.0175e-014
```

由前面给出的结果可见：如果对一个矩阵的广义逆再求一次广义逆，则将还原成原来的矩

阵，亦即 $(\boldsymbol{A}^+)^+ = \boldsymbol{A}$。

3.2.5.3 Kronecker 积与矩阵方程求解

考虑一类线性代数方程

$$\boldsymbol{AX} = \boldsymbol{C} \tag{3.38}$$

其中 $\boldsymbol{A}$ 为 $n \times n$ 矩阵，且 $\boldsymbol{C}$ 为 $n \times m$ 矩阵，为方便叙述可以将上面各个矩阵的参数记成

$$\boldsymbol{X}=\begin{bmatrix} x_1 & x_2 & \cdots & x_m \\ x_{m+1} & x_{m+2} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{(n-1)m+1} & x_{(n-1)m+2} & \cdots & x_{nm} \end{bmatrix}, \boldsymbol{C}=\begin{bmatrix} c_1 & c_2 & \cdots & c_m \\ c_{m+1} & c_{m+2} & \cdots & c_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ c_{(n-1)m+1} & c_{(n-1)m+2} & \cdots & c_{nm} \end{bmatrix} \tag{3.39}$$

该方程的解仍可以由 $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{C}$ 求出，同时由 MATLAB 函数可以直接求出该方程的解 `X = inv(A)*C`。在很多应用中，人们更希望将上面的方程转换成方程右边为一个列向量，且方程的解也由一个列向量来表示的形式，这需要进行特殊的变换。可以证明，该方程可以变换成

$$(\boldsymbol{A} \otimes \boldsymbol{I}_m)\boldsymbol{x} = \boldsymbol{c} \tag{3.40}$$

式中 $\otimes$ 表示两个矩阵的 Kronecker 乘积，而 $\boldsymbol{x}$ 和 $\boldsymbol{c}$ 分别为列向量，其表示方法为

$$\boldsymbol{x}^T = [x_1 \ x_2 \ \cdots \ x_{nm}], \ \boldsymbol{c}^T = [c_1 \ c_2 \ \cdots \ c_{nm}] \tag{3.41}$$

这样原方程的解 $\boldsymbol{x}$ 就可以容易地求出了。

上面演示的算法并不是利用 Kronecker 乘积的主要目的，理解了上述的变换之后，还可以用 Kronecker 乘积来处理更复杂的方程，比如下面给出的 Lyapunov 方程

$$\boldsymbol{AX} + \boldsymbol{XB} = \boldsymbol{C} \tag{3.42}$$

式中 $\boldsymbol{A}$ 为 $n \times n$ 矩阵，$\boldsymbol{B}$ 为 $m \times m$ 矩阵。利用 Kronecker 乘积的表示方法，上面的方程可以写成

$$(\boldsymbol{A} \otimes \boldsymbol{I}_m + \boldsymbol{I}_n \otimes \boldsymbol{B}^T)\boldsymbol{x} = \boldsymbol{c} \tag{3.43}$$

〖例 3.14〗 假设式 (3.42) 中的 $\boldsymbol{A}, \boldsymbol{B}$ 和 $\boldsymbol{C}$ 矩阵分别为

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}, \quad \boldsymbol{B} = \boldsymbol{A}^T, \quad \boldsymbol{C} = \begin{bmatrix} 1 & 5 & 4 \\ 5 & 6 & 7 \\ 4 & 7 & 9 \end{bmatrix}$$

则可以由下面的 MATLAB 语句求出该方程的解

```
>> A=[1 2 3;4 5 6; 7 8 0];  B=A'; C=[1, 5, 4; 5, 6, 7; 4, 7, 9];
   A0=kron(A,eye(3))+kron(eye(3),B')
A0 =
```

```
     2     2     3     2     0     0     3     0     0
     4     6     6     0     2     0     0     3     0
     7     8     1     0     0     2     0     0     3
     4     0     0     6     2     3     6     0     0
     0     4     0     4    10     6     0     6     0
     0     0     4     7     8     5     0     0     6
     7     0     0     8     0     0     1     2     3
     0     7     0     0     8     0     4     5     6
     0     0     7     0     0     8     7     8     0
>> c=[C(1,:) C(2,:) C(3,:)];  x0 = inv(A0)*c'
x0 =
    1.5556
   -1.1111
    0.3889
   -1.1111
    1.2222
    0.2222
    0.3889
    0.2222
    0.3889
>> D=[x0(1:3)'; x0(4:6)'; x0(7:9)']
D =
    1.5556   -1.1111    0.3889
   -1.1111    1.2222    0.2222
    0.3889    0.2222    0.3889
>> C0=A*D+D*A'
C0 =
    1.0000    5.0000    4.0000
    5.0000    6.0000    7.0000
    4.0000    7.0000    9.0000
>> norm(C0-C)
ans =
    9.3820e-015
```

上面求出了需要的 Lyapunov 方程的解，并将该解代入了原始的方程中，可见得出的结果满足原始方程，其误差也非常小，故该算法可以由相当高的精度求解相应的矩阵方程。

MATLAB 提供了一个 reshape() 函数来改变矩阵的维数，该函数用于此问题中就很贴切，使用该函数则可以将前面的相应语句修改成

```
>> c=reshape(C,9,1); x0=inv(A0)*c; D=reshape(x0,3,3);
```

```
D =
    1.5556   -1.1111    0.3889
   -1.1111    1.2222    0.2222
    0.3889    0.2222    0.3889
```

事实上，MATLAB 的控制系统工具箱中也提供了 Lyapunov 方程的求解函数 lyap()，这里直接使用该函数得出方程的解

```
>> X=lyap(A,A',-C)
X =
    1.5556   -1.1111    0.3889
   -1.1111    1.2222    0.2222
    0.3889    0.2222    0.3889
```

将得出的解回代到原方程中则有

```
>> norm(A*X+X*A'-C)
ans =
    2.1970e-014
```

可见直接调用 lyap() 函数得出的方程解和前面的方法得出的是几乎完全一致的，两者之间有差异是因为所采用的算法不同，对此例来说似乎由 Kronecker 积的算法得出解的精度稍高一些，因为该算法只涉及矩阵求逆，传递的误差较小。

3.2.6 矩阵的非线性运算

3.2.6.1 面向矩阵各个元素的非线性运算

MATLAB 提供了大量函数，允许用户对矩阵进行处理，前面介绍的主要是矩阵的线性变换，本节将介绍如何对矩阵进行非线性运算。

事实上，MATLAB 提供了两类函数，其中一类是对矩阵的各个元素进行单独运算的，而后一类是对整个矩阵进行运算的。前面曾经用到了 sin() 函数，该函数属于第一类，是对矩阵的各个元素单独运算的，而不是对整个矩阵进行运算的。这类常用的 MATLAB 函数在表 3-2 中列出来，它们的调用方法是很显然的，其标准调用格式为：

```
B = 函数名(A); 例如 B=sin(A);
```

表 3-2 面向矩阵元素的非线性函数表

函数名	意　义	函数名	意　义
abs()	求模（绝对值）函数	asin(), acos()	反正弦、余弦函数
sqrt()	求平方根函数	log(), log10()	自然和常用对数
exp()	指数函数	real(), imag(), conj()	求实虚部及共轭复数
sin(), cos()	正弦、余弦函数	round(), floor(), ceil()	取整数函数

〖例 3.15〗 考虑例 3.1 中给出的 $\boldsymbol{A}$ 矩阵，调用其中的一些函数，其结果在下面给出。

```
>> A=[1,2,3; 4,5,6; 7,8,0]; exp(A)
ans =
   1.0e+003 *
   0.0027    0.0074    0.0201
   0.0546    0.1484    0.4034
   1.0966    2.9810    0.0010
>> sin(A)
ans =
    0.8415    0.9093    0.1411
   -0.7568   -0.9589   -0.2794
    0.6570    0.9894         0
>> log(A)
Warning: Log of zero.
ans =
         0    0.6931    1.0986
    1.3863    1.6094    1.7918
    1.9459    2.0794      -Inf
>> log10(A)
Warning: Log of zero.
ans =
         0    0.3010    0.4771
    0.6021    0.6990    0.7782
    0.8451    0.9031      -Inf
>> B = 10*inv(A)
B =
  -17.7778    8.8889   -1.1111
   15.5556   -7.7778    2.2222
   -1.1111    2.2222   -1.1111
>> round(B)
ans =
   -18     9    -1
    16    -8     2
    -1     2    -1
```

3.2.6.2 面向整个矩阵的非线性运算

除了对矩阵的单个元素进行单独计算以外，一般还常常要求对整个矩阵做这样的非线性运算。例如想求出一个矩阵的 e 指数，就需要特殊的算法来完成了。文献 [36]

中叙述了求解矩阵指数的 19 种不同方法，每一种方法都有自己的特点及适用范围。在 MATLAB 下提供了 4 个求取矩阵指数的函数：expm(), expm1(), expm2() 和 expm3()，其中的 expm() 为内在函数，它采用 Padé 近似技术来求取矩阵的指数，而 expm1() 函数是 expm() 函数的 M 函数实现。函数 expm2() 采用 Taylor 级数展开方法来求取矩阵的指数，该方法比较直观，直接对矩阵指数作幂级数展开。

$$\mathrm{e}^{\boldsymbol{A}}=\sum_{i=0}^{\infty}\frac{1}{i!}\boldsymbol{A}^i=\boldsymbol{I}+\boldsymbol{A}+\frac{1}{2}\boldsymbol{A}^2+\frac{1}{3!}\boldsymbol{A}^3+\cdots+\frac{1}{m!}\boldsymbol{A}^m+\cdots \tag{3.44}$$

可以看出，这样的运算可以由 while 循环结构来编程，当幂级数累加项的范数满足误差要求时退出循环即可。

expm3() 采用特征值特征向量的方法求出矩阵的指数矩阵，其数学原理如下：首先求出矩阵 $\boldsymbol{A}$ 的特征值 $\boldsymbol{D}=\mathrm{diag}(\gamma_1,\gamma_2,\cdots,\gamma_n)$ 及相应的特征向量矩阵 $\boldsymbol{V}$，然后对该对角矩阵求取矩阵指数，亦即对每个对角矩阵元素求指数，这时原矩阵 $\boldsymbol{A}$ 的指数矩阵为

$$\mathrm{e}^{\boldsymbol{A}}=\boldsymbol{V}\begin{bmatrix}\mathrm{e}^{\gamma_1} & & \\ & \ddots & \\ & & \mathrm{e}^{\gamma_n}\end{bmatrix}\boldsymbol{V}^{-1} \tag{3.45}$$

这种方法看似简单，但有很大的局限性，它一般要求原矩阵没有重根，否则往往得出的特征向量矩阵趋于奇异，因而可能得出错误的结果，下面将给出演示例子。

【例 3.16】 *考虑下面给出的矩阵* $\boldsymbol{A}$

$$\boldsymbol{A}=\begin{bmatrix}-2 & 1 & 0 & & \\ 0 & -2 & 1 & & \\ 0 & 0 & -2 & & \\ & & & -5 & 1 \\ & & & 0 & -5\end{bmatrix}$$

如果对此矩阵进行指数运算，则可以获得以下的结果

```
>> A=[[-2 1 0; 0 -2 1; 0 0 -2], zeros(3,2); zeros(2,3) [-5 1; 0 -5]];
   expm(A)
ans =
     0.1353    0.1353    0.0677         0         0
          0    0.1353    0.1353         0         0
          0         0    0.1353         0         0
          0         0         0    0.0067    0.0067
          0         0         0         0    0.0067
>> logm(ans)
ans =
    -2.0000    1.0000    0.0000   -0.0000    0.0000
     0.0000   -2.0000    1.0000    0.0000    0.0000
    -0.0000   -0.0000   -2.0000    0.0000    0.0000
```

```
      0.0000    0.0000   -0.0000   -5.0000    1.0000
      0.0000    0.0000   -0.0000    0.0000   -5.0000
>> norm(ans-A)
ans =
      2.6762e-014
```

对得出的指数结果进行对数运算可以按相当高的精度还原原始矩阵，从而表明，矩阵对数运算还是很精确。

事实上，因为原来的矩阵 $\boldsymbol{A}$ 为一个标准的 Jordan 矩阵，所以，如果对两个 Jordan 子矩阵作指数运算，则也可以得出该矩阵的指数矩阵。

```
>> B=[expm(A(1:3,1:3)), zeros(3,2); zeros(2,3), expm(A(4:5,4:5))]
B =
    0.1353    0.1353    0.0677         0         0
         0    0.1353    0.1353         0         0
         0         0    0.1353         0         0
         0         0         0    0.0067    0.0067
         0         0         0         0    0.0067
```

可见两种方法得出的指数矩阵是相同的。如果采用 expm2() 函数来求取矩阵指数，则可以发现要经过幂级数的 41 步累加，且其累加项的范数可达 9.0548×10^{-20}，显见这一函数可以按足够的精度得出正确的结果。对同样一个矩阵调用 expm3() 函数来求取矩阵指数，则可以得出

```
>> expm3(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.972152e-031.
> In C:\MATLAB6p1\toolbox\matlab\demos\expm3.m at line 13
ans =
    0.1353         0         0         0         0
         0    0.1353         0         0         0
         0         0    0.1353         0         0
         0         0         0    0.0067         0
         0         0         0         0    0.0067
```

其中给出了警告信息，通知用户得出的特征向量矩阵为坏条件矩阵 (条件数的倒数的估计值达 10^{-39})。分析一下原矩阵的特征值

```
>> eig(A)'
ans =
      -2    -2    -2    -5    -5
```

可以看出，该矩阵有两组重根：-2 (3 重) 和 -5 (2 重)，这将使得得出的特征向量矩阵出现奇异现象，因而导致得出错误的矩阵指数。所以在使用 expm3() 函数时需要引起足够的重视，以免得出错误的结果。

除了对整个矩阵求取矩阵指数之外，MATLAB 还允许对矩阵进行其他非线性变换，

其中常用的函数还有 logm() (矩阵求对数)、sqrtm() (矩阵求平方根) 和 funm() (矩阵求任意函数) 等。

可以看出，这里的函数名很有特点，每个函数名在标准函数名的后面加了一个后缀 m，表示对矩阵而不是对矩阵元素进行运算。这里的 funm() 函数可以求出矩阵的任意函数，其调用方法为 funm(矩阵名，'函数名')，其中函数名应该由单引号括起来，例如若想求出矩阵 $\boldsymbol{A}$ 的正弦矩阵，则可以使用如下的命令 B = funm(A,'sin')。值得指出的是，这里给出的矩阵函数运算是基于矩阵特征值特征向量而完成的，类似于 expm3() 的效果，故在一些特殊但很常见的场合下仍将出现错误。

重新考虑前面的例子，如果想对其中的 $\boldsymbol{A}$ 矩阵进行正弦运算，则将得出如下的错误结论：

```
>> funm(A,'sin')
WARNING: Result from FUNM may be inaccurate. esterr = 1
ans =
   -0.9093         0         0         0         0
         0   -0.9093         0         0         0
         0         0   -0.9093         0         0
         0         0         0    0.9589         0
         0         0         0         0    0.9589
```

事实上矩阵的非线性函数运算可以通过幂级数的方法简单地求出，例如正弦函数可以由下面的幂级数展开式求出

$$\sin \boldsymbol{A}=\sum_{i=0}^{\infty}(-1)^i\frac{\boldsymbol{A}^{2i+1}}{(2i+1)!}=\boldsymbol{A}-\frac{1}{3!}\boldsymbol{A}^3+\frac{1}{5!}\boldsymbol{A}^5+\cdots \tag{3.46}$$

可以用 MATLAB 实现正弦函数幂级数的展开

```
function E=sinm1(A)
E = zeros(size(A)); F = A; k = 1;
while norm(E+F-E,1) > 0
   E = E + F;  F = -A^2*F/((k+2)*(k+1)); k = k+2;
end
```

由上面的程序可以看出，看起来比较复杂的矩阵正弦函数的幂级数展开运算可以由几条 MATLAB 语句容易地编写出来。在习题中给出了余弦及反正弦函数的幂级数展开公式，用户可以根据这些公式，并仿照前面给出的程序段很简洁地写出相应的程序，则可以容易地求出原矩阵 $\boldsymbol{A}$ 的正弦矩阵为

```
>> E=sinm1(A)
E =
   -0.9093   -0.4161    0.4546         0         0
         0   -0.9093   -0.4161         0         0
         0         0   -0.9093         0         0
         0         0         0    0.9589    0.2837
```

```
        0         0         0         0    0.9589
```

可以测出，该函数一共进行了 39 次叠加运算。对上面的结果矩阵再进行反正弦运算，不难得出这样的结论：这种运算可以还原出原来的矩阵 $\boldsymbol{A}$，而这样的结果光靠 MATLAB 提供的 funm() 函数是不可能得出的。所以在使用 funm() 函数时应该格外注意，如果确实不能得出正确的结果，则建议采用幂级数的方法编写程序来直接求出。

3.2.7 线性代数问题的解析求解

MATLAB 语言不但提供了丰富的线性代数问题数值解的求解函数，还和著名的符号运算语言 Maple 有机结合，并以其为符号运算引擎提供了符号运算工具箱，该工具箱中的函数可以用来求解线性代数问题的解析解。

在使用符号运算工具箱之前，需要把一些变量声明为“符号变量”，以区别于常规的数值变量。将 x 变量声明符号变量的语句格式为：

```
x=sym('x',变量其他说明)
```

其中，“变量其他说明”选项包括实数 'real'、正数 'positive' 等，对一般变量没有必要声明这样的类型。矩阵变量可以由 sym(A) 语句声明。还可以使用 syms 命令同时声明若干个变量及其类型，如：

```
>> syms a b c d  % 声明 a,b,c,d 为符号变量，注意要用空格分隔变量名
   syms a c b positive % 将 a,b,c 设置成正的符号变量
```

下面将通过例子演示线性代数问题的解析解法。

〖例 3.17〗 构造一个 5 阶的 Hilbert 矩阵，可以求出其逆、行列式、特征多项式、特征值等常用矩阵分析的解析解：

```
>> H=sym(hilb(5)) % 构造并显示 5 阶 Hilbert 矩阵
H =
[  1, 1/2, 1/3, 1/4, 1/5]
[ 1/2, 1/3, 1/4, 1/5, 1/6]
[ 1/3, 1/4, 1/5, 1/6, 1/7]
[ 1/4, 1/5, 1/6, 1/7, 1/8]
[ 1/5, 1/6, 1/7, 1/8, 1/9]
>> B=inv(H)    % 计算逆矩阵解析解
B =
[      25,    -300,    1050,   -1400,     630]
[    -300,    4800,  -18900,   26880,  -12600]
[    1050,  -18900,   79380, -117600,   56700]
[   -1400,   26880, -117600,  179200,  -88200]
[     630,  -12600,   56700,  -88200,   44100]
>> D=det(H)    % 矩阵的行列式解析解
```

```
D =
    1/266716800000
>> P=poly(H); pretty(P) % 计算并以可读的形式显示特征多项式
     5   563  4    735781    3     852401     2       61501
    x  - --- x  + ------- x  - --------- x  + ----------- x - 1/266716800000
         315      2116800      222264000      53343360000
>> eig(H) % 矩阵的特征值高精度解
ans =
[ .3287928772171862957115004760544 8e-5]
[ .3058980401511917268794978406927 2e-3]
[ .1140749162341980655945145886658 9e-1]
[     .2085342186110133359050025100688 2]
[     1.5670506910982307955330110055207]
```

事实上，这里得出的特征值是高精度的数值解，因为由著名的 Abel 定理可知，5 阶及 5 阶以上的一般代数方程是没有解析解的。

〖例 3.18〗 考虑例 3.16 中的 5 阶 Jordan 块矩阵，可以用下面语句求出其状态转移矩阵 e^{At}

```
>> A=[[-2 1 0; 0 -2 1; 0 0 -2], zeros(3,2); zeros(2,3) [-5 1; 0 -5]];
   syms t; B=expm(A*t)
B =
[ exp(-2*t), t*exp(-2*t), 1/2*t^2*exp(-2*t),         0,           0]
[         0,   exp(-2*t),       t*exp(-2*t),         0,           0]
[         0,           0,         exp(-2*t),         0,           0]
[         0,           0,                 0, exp(-5*t), t*exp(-5*t)]
[         0,           0,                 0,         0,   exp(-5*t)]
```

值得说明的是，由于这里涉及的是 5 阶特殊矩阵，所以可以精确地求出其状态转移矩阵，对一般的高阶矩阵来说，求解出来的状态转移矩阵的可读性不是很高。

3.3 微积分问题的 MATLAB 求解

3.3.1 数值差分与微分运算

3.3.1.1 数值差分运算

MATLAB 语言提供了计算给定向量差分的函数 diff()，其调用方法是很直观的：dy = diff(y)。假设向量 $\boldsymbol{y}$ 是由 $\{y_i\}, i=1,2,\cdots,n$ 构成的，则经 diff() 函数处理后将得出一个新的向量：$\{y_{i+1}-y_i\}, i=1,2,\cdots,n-1$，显然新得出向量 $\Delta\boldsymbol{y}$ 的长度比原向量 $\boldsymbol{y}$ 的长度小 1。

〖例 3.19〗 如果原向量 $\boldsymbol{y}$ 为行向量，则 $\Delta\boldsymbol{y}$ 向量也是行向量；若 $\boldsymbol{y}$ 为列向量，则 $\Delta\boldsymbol{y}$ 向量也是列向量。MATLAB 提供的函数 diff() 还可以扩展到矩阵和多维数组，考虑如下构造的

Vandermonde 矩阵

```
>> V=vander(1:6),
V =
       1       1       1       1       1       1
      32      16       8       4       2       1
     243      81      27       9       3       1
    1024     256      64      16       4       1
    3125     625     125      25       5       1
    7776    1296     216      36       6       1
```

还可以直接使用 diff() 对之进行差分运算

```
>> diff(V)
ans =
      31      15       7       3       1       0
     211      65      19       5       1       0
     781     175      37       7       1       0
    2101     369      61       9       1       0
    4651     671      91      11       1       0
```

可以看出，diff()函数对矩阵的每一列都进行差分运算，故而结果矩阵的列数是不变的，只有行数减 1。事实上，MATLAB 下的很多函数都是按列进行操作的，如后面要介绍的 mean() 函数等。

MATLAB 语言提供的 **gradient()** 函数可以直接用来求取一个矩阵的二维差分，该函数的调用格式为：

```
[dx, dy]=gradient(A)
```

【例 3.20】 考虑前面的 Vandermonde 矩阵，可以由下面的命令求出其二维差分。

```
>> [dx,dy]=gradient(V)
dx =
  1.0e+003 *
         0         0         0         0         0         0
   -0.0160   -0.0120   -0.0060   -0.0030   -0.0015   -0.0010
   -0.1620   -0.1080   -0.0360   -0.0120   -0.0040   -0.0020
   -0.7680   -0.4800   -0.1200   -0.0300   -0.0075   -0.0030
   -2.5000   -1.5000   -0.3000   -0.0600   -0.0120   -0.0040
   -6.4800   -3.7800   -0.6300   -0.1050   -0.0175   -0.0050
dy =
      31      15       7       3       1       0
     121      40      13       4       1       0
     496     120      28       6       1       0
```

```
1441    272    49     8    1    0
3376    520    76    10    1    0
4651    671    91    11    1    0
```

3.3.1.2 数值微分算法

常用的数值微分算法包括前向差分算法、后向差分算法和中心差分算法。

- 前向差分算法的一阶微分表达式为

$$y_i' = \frac{\Delta y_i}{\Delta t} = \frac{y_{i+1} - y_i}{\Delta t} \tag{3.47}$$

- 后向差分算法的一阶微分表达式为

$$y_i' = \frac{\Delta y_i}{\Delta t} = \frac{y_i - y_{i-1}}{\Delta t} \tag{3.48}$$

这两种微分算法的精度都是 $o(\Delta t)$ 级的。经实践检验，利用基于前向和后向差分的数值微分算法求取高阶微分时的精度一般都是很低的，所以这里只介绍两种中心差分的算法。首先定义一阶微分为

$$y_i' = \frac{\Delta y_i}{\Delta t} = \frac{y_{i+1} - y_{i-1}}{2\Delta t} \tag{3.49}$$

记

$$\tilde{f}'(x) = \frac{f(x+\Delta t) - f(x-\Delta t)}{2\Delta t} \tag{3.50}$$

由 Taylor 级数展开可以将上式进一步写成

$$\begin{aligned}\tilde{f}(x) = &\frac{f(x) + \Delta t f'(x) + \Delta t^2 f''(x)/2! + \Delta t^3 f''(\xi)/3!}{2\Delta t} - \\ &\frac{f(x) - \Delta t f'(x) + \Delta t^2 f''(x)/2! - \Delta t^3 f''(\xi)/3!}{2\Delta t} = f'(x) + \frac{\Delta t^3}{3!} f''(\xi)\end{aligned} \tag{3.51}$$

可见这种中心差分的算法精度为 $o(\Delta t^2)$。该中心差分算法的高阶微分公式为

$$\begin{aligned}y_i'' &= \frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta t^2} \\ y_i''' &= \frac{y_{i+2} - 2y_{i+1} + 2y_{i-1} - y_{i-2}}{2\Delta t^3} \\ y_i^{(4)} &= \frac{y_{i+2} - 4y_{i+1} + 6y_i - 4y_{i-1} + y_{i-2}}{\Delta t^4}\end{aligned} \tag{3.52}$$

另一种具有 $o(\Delta t^4)$ 精度级的中心差分算法为

$$\begin{aligned}y_i' &= \frac{-y_{i+2} + 8y_{i+1} - 8y_{i-1} + y_{i-2}}{12\Delta t} \\ y_i'' &= \frac{-y_{i+2} + 16y_{i+1} - 30y_i + 16y_{i-1} - y_{i-2}}{12\Delta t^2} \\ y_i''' &= \frac{-y_{i+3} + 8y_{i+2} - 13y_{i+1} + 13y_{i-1} - 8y_{i-2} + y_{i-3}}{8\Delta t^3} \\ y_i^{(4)} &= \frac{-y_{i+3} + 12y_{i+2} - 39y_{i+1} + 56y_i - 39y_{i-1} + 12y_{i-2} - y_{i-3}}{6\Delta t^4}\end{aligned} \tag{3.53}$$

根据上述的算法，可以写出中心差分算法求取数值微分的函数 diffctr()

```
function dy = diffctr(y, h, n, key)
yx1=[y 0 0 0 0 0]; yx2=[0 y 0 0 0 0]; yx3=[0 0 y 0 0 0];
yx4=[0 0 0 y 0 0]; yx5=[0 0 0 0 y 0]; yx6=[0 0 0 0 0 y];
switch n
case 1
   if key==1,dy=(diff(yx1)+diff(yx2))/(2*h); L0=3;
   else,
      dy = (-diff(yx1)+7*diff(yx2)+7*diff(yx3)-...
            diff(yx4))/(12*h); L0=4;
   end
case 2
   if key==1, dy=(diff(yx1)-diff(yx2))/(h^2); L0=3;
   else,
      dy=(-diff(yx1)+15*diff(yx2)-15*diff(yx3)+...
            diff(yx4))/(12*h^2);L0=4;
   end
case 3
   if key==1,
      dy=(diff(yx1)-diff(yx2)-diff(yx3)+diff(yx4))/(2*h^3); L0=4;
   else,
      dy=(-diff(yx1)+7*diff(yx2)-6*diff(yx3)-6*diff(yx4)+...
         7*diff(yx5)-diff(yx6))/(8*h^3); L0=5;
   end
case 4
   if key==1,
      dy=(diff(yx1)-3*diff(yx2)+3*diff(yx3)-diff(yx4))/(h^4); L0=4;
   else,
      dy = (-diff(yx1)+11*diff(yx2)-28*diff(yx3)+28*diff(yx4)-...
         11*diff(yx5)+diff(yx6))/(6*h^4);L0=5;
   end
end
L1=L0; if key==2, L1=L1+1; end, dy=dy(L1:end-L0);
```

该函数的调用格式为：

```
dy = diffctr(y, h, n, key)
```

其中，y为给定的一组数值；h 为 y 数据的时间间隔；n 为想求出微分的阶次，该函数中实现了 1 到 4 阶的微分；key 为选定的数值微分算法。有了这些参数，则所得出的 n 阶数

值微分结果将由 dy 返回。

〖例 3.21〗 假设有原始函数

$$f(x)=\frac{\sin x}{x+\cos 2x}$$

取步长 $h=0.05$，则可以得出各个采样点的值，依据该采样点的值。调用前面的数值微分函数，则可以得出原函数的 1 到 4 阶数值微分。

```
>> h=0.05; x=0:h:pi; y=sin(x)./(x+cos(2*x));
   y11=diffctr(y,h,1,1);  y12=diffctr(y,h,1,2);
   y21=diffctr(y,h,2,1);  y22=diffctr(y,h,2,2);
   y31=diffctr(y,h,3,1);  y32=diffctr(y,h,3,2);
   y41=diffctr(y,h,4,1);  y42=diffctr(y,h,4,2);
```

事实上，由该函数可以解析地求出其 1 到 4 阶导数。由于高阶导数表达式很复杂，所以这里只列出一阶导数

$$f'(x)=\frac{\cos x}{x+\cos(2x)}-\frac{\sin x(1-2\sin(2x))}{(x+\cos(2x))^2}$$

高阶导数的理论值可以从下面的语句中计算出来，推导过程从略。

```
>> D=x+cos(2*x); DD=D.^2; DDD=D.^3; D4=D.^4;
   E=1-2*sin(2*x); EE=E.^2; EEE=E.^3; E4=E.^4;
   yy1=diff(y)/h; f1=cos(x)./D-sin(x).*E./DD;
   f2=4*cos(2*x).*sin(x)./DD-y-2*cos(x).*E./DD+2*sin(x).*EE./DDD;
   f3=12*cos(x).*cos(2*x)./DD-cos(x)./D-24*cos(2*x).*y.*E./DD+...
      3*y.*E./D+6*cos(x).*EE./DDD-6*y.*EEE./DDD-8*y.*sin(2*x)./D;
   f4=96*cos(2*x).^2.*y./DD+y-40*cos(2*x).*y./D+4*cos(x).*E./DD-...
      96*cos(x).*cos(2*x).*E./DDD+4*cos(x).*E./DD+...
      144*cos(2*x).*y.*EE./DDD-12*y.*EE./DDD-24.*cos(x).*EEE./D4+...
      24*y.*E4./D4-32*cos(x).*sin(2*x)./DD+64*y.*E.*sin(2*x)./DD;
```

由下面的语句可以将各阶微分的结果和理论曲线都绘制出来，如图 3-1 所示。

```
>> L=length(y); i1=3:L; i2=4:L-2; i3=3:L-2; i4=4:L-4;
   subplot(221),plot(x,f1,x(i1),y11(1:end-1),'--',x(i2),y12(1:end-1),':')
   subplot(222),plot(x,f2,x(i1),y21(1:end-1),'--',x(i2),y22(1:end-1),':')
   subplot(223),plot(x,f3,x(i3),y31(1:end-1),'--',x(i4),y32(1:end-1),':')
   subplot(224),plot(x,f4,x(i3),y41(1:end-1),'--',x(44),y42(1:end-1),':')
```

可见，一阶、二阶微分值和理论值极其接近，在曲线上不能区分出来，三阶和四阶数值微分在特定的位置上略有误差，但误差是很小的。故中心差分算法的精度还是很高的，所以在实际应用中可以采用此函数。

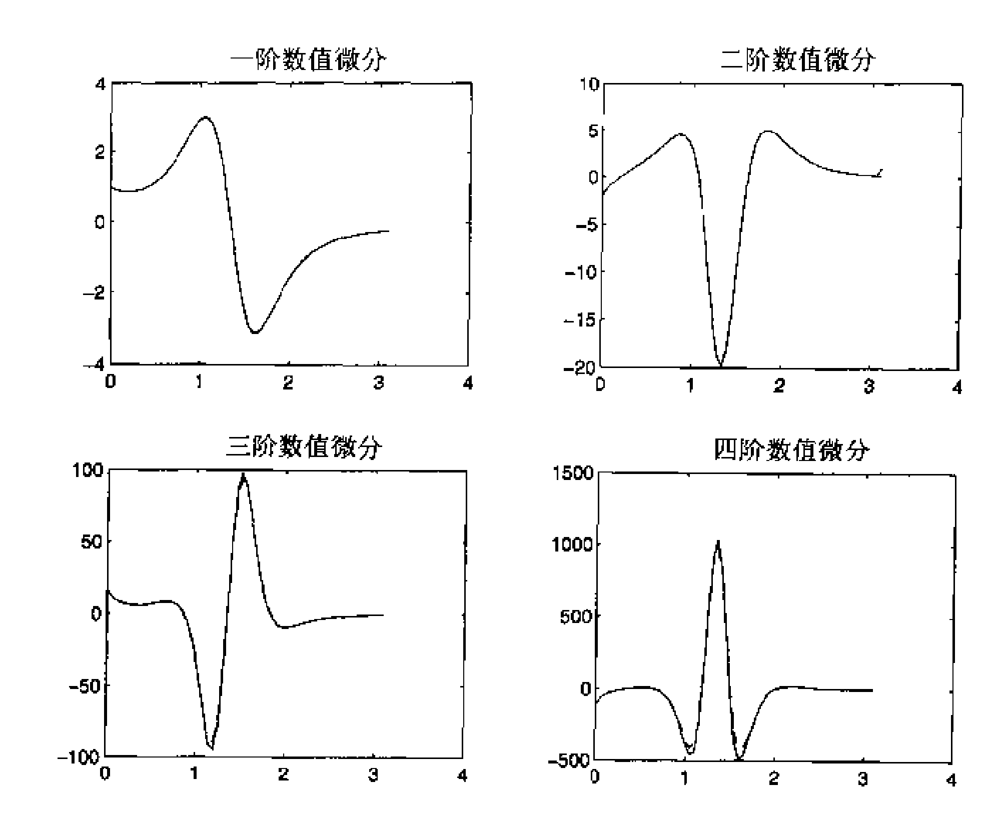


图 3-1 中心差分算法的精度演示

3.3.2 数值积分运算

考虑一维函数的定积分

$$I=\int_a^b f(x)\mathrm{d}x \tag{3.54}$$

在被积函数 $f(x)$ 相当复杂时，即使有强大的计算机代数系统帮忙，也不一定能求出解析解，或解析解根本不存在，所以往往要采用数值方法来求解。求解定积分的数值方法是多种多样的，如简单的梯形法、Simpson 法、Romberg 法等都是经常采用的方法。它们的基本思想都是将整个积分空间 $[a,b]$ 分割成若干个子空间 $[x_i,x_{i+1}]$, $i=1,2,\cdots,N$，其中 $x_1=a$, $x_{N+1}=b$。这样整个积分问题就分解为下面的求和形式

$$\int_a^b f(x)\mathrm{d}x=\sum_{i=1}^N\int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x \tag{3.55}$$

而在每一个小的子空间上都可以近似地求解出来，例如可以采用下面给出的 Simpson 方法来求解出 $[x_i,x_{i+1}]$ 上的积分近似值

$$\int_{x_i}^{x_{i+1}} f(x)\mathrm{d}x\simeq\frac{h_i}{12}\left[f(x_i)+4f\left(x_i+\frac{h_i}{4}\right)+2f\left(x_i+\frac{h_i}{2}\right)+4f\left(x_i+\frac{3h_i}{4}\right)+f(x_i+h_i)\right] \tag{3.56}$$

式中 $h_i=x_{i+1}-x_i$。MATLAB 基于此算法，采用自适应变步长方法给出了 `quad()` 函数

来求取定积分，该函数的调用格式为：

```
[y,n]=quad(F, a, b, tol);
```

其中 F 为描述被积函数的字符串变量，一般为一个 F.m 函数文件名，该函数的一般格式为 y= F(x); a, b 分别为定积分的上限和下限；tol 为变步长积分用的误差限，如果用户不给出误差限，则将自动地假定为默认值 tol=1e-3。返回的 n 为被积行数的调用次数，在实际应用中若不关心被积函数的调用次数，则可以略去该变量。

〖例 3.22〗 试求出下面的无穷定积分

$$\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\mathrm{e}^{-x^2/2}\mathrm{d}x$$

从高等数学中可知，该不定积分是没有解析解的，而无穷定积分的理论值为 1。这一无穷定积分可以由有穷的积分来近似，一般情况下，选择积分的上下限为 ±8 就能保证相当的精度。由给出的被积函数可以立即写出下面的函数：

```
function y=myerrf(x)
y=1/sqrt(2*pi)*exp(-x.^2/2);
```

假定这时可以通过下面的 MATLAB 语句求出所需函数的定积分

```
>> format long; [y,kk]=quad('myerrf',-8,8),
y =
   0.99999997742635
kk =
   504
```

增大积分的上下限，则可以给出如下的 MATLAB 命令：

```
>> [y,kk]=quad('myerrf',-15,15),
Warning: Recursion level limit reached in quad. Singularity likely.
y =
   1.00000007062479
kk =
   860
```

对本问题来说，如果将积分上下限选择得过大，则可能出现警告信息；其实这一算法的精度不是很高，且需要调用 504 次 myerrf() 函数。

在 MATLAB 6.0 中还引入了"函数句柄"的概念，其函数编写的格式与前面介绍的完全一致，在使用该函数的时候，则不用引号括起函数名，而直接用由 @ 符号引导的函数句柄即可。

除了直接以文件形式写出一个 M 函数来描述原问题之外，还允许用 inline() 函数来定义一个函数，这样就使得问题更加简化了。例如本例中的问题还可以如下求解：

```
>> f=inline('1/sqrt(2*pi)*exp(-x.^2/2)','x'),
   [y,kk]=quad(f,-8,8),
y =
   0.99999997742635
```

```
kk =
    860
```

可见，采用 inline() 函数描述模型更为方便，因为这样做可以少建立没必要的附加文件。

文献 [9] 给出了一种高精度的算法，该算法利用了插值运算来更精确更快速地求出所需要的定积分。该算法又称为 Newton Cotes 方法。例如，在 $[0,1]$ 区间上的积分可以近似地表示成

$$\int_0^1 f(x)\mathrm{d}x \simeq \sum_{k=0}^{8} w_k f\left(\frac{k}{8}\right) \tag{3.57}$$

其中加权系数 w_k 为

$$w_0=\frac{989}{28350},\ w_1=\frac{2944}{14175},\ w_2=-\frac{4644}{14175},\ w_3=\frac{5248}{14175}, w_4=-\frac{454}{2835} \tag{3.58}$$

且 $w_{8,7,6,5}=w_{0,1,2,3}$，这些加权系数满足

$$\sum_{k=0}^{8} w_k=1,\ \sum_{k=0}^{8} \mid w_k \mid = 1.4512 \tag{3.59}$$

在这一算法下定积分求值的 MATLAB 实现在函数 quad8() 中给出，该函数的调用格式为：

```
[y,n]=quad8(F, a, b, tol);
```

该函数的调用格式和 quad() 完全一致，而函数中 tol 的默认值为 10^{-6}。该算法可以更精确地求出积分的值，且一般情况下函数调用的次数明显小于 quad()，故而保证能以更高的效率求解出所需的定积分值。

对于上面同样的积分使用 quad8()，则可以使用更大的积分上下限来逼近给出的 $(-\infty,\infty)$ 空间，所以，选择积分限为 $[-15,15]$，则可以通过下面的命令来求解积分问题

```
>> [y,kk]=quad8(@myerrf,-15,15),
y =
    0.99999999999999
kk=
    96
>> [y,kk]=quad8(@myerrf,-8,8),
y =
   1.00000000000023
kk =
    20
```

该函数仅调用了 96 次 myerrf() 函数，而同样在 $[-8,8]$ 区间内进行积分运算只需调用 20 次被积函数，因而其效率要明显高于 quad() 函数，精度也高得多，且不会出现像 quad() 函数调用时那样的错误信息提示。

MATLAB 新引进的数值积分函数 quadl() 是基于 Lobatto 算法编写的，其调用格式与 quad8() 完全一致，可以试验用该方法解决前面的问题。

在前面的语句中如果用quadl() 函数直接取代 quad8()，则将得出如下的结果

```
>> [y,kk]=quadl(f,-15,15),
y =
   1.00000000000338
kk =
   288
>> [y,kk]=quadl(f,-8,8),
y =
   0.99999999999993
kk =
   168
```

可以看出，对给出的问题来说，用新函数求解的精度要远远低于传统的 quad8() 函数，其被积函数的调用次数也明显增多，所以采用新函数未必能改进计算效果。

3.3.3 多重定积分的数值求解

考虑下面的双重定积分问题

$$I = \int_{y_m}^{y_M} \int_{x_m}^{x_M} f(x,y)\mathrm{d}x\mathrm{d}y \tag{3.60}$$

使用 MATLAB 提供的 dblquad() 函数就可以直接求出上述双重定积分的数值解。该函数的调用格式为：

```
y=dblquad(函数名, x_m, x_M, y_m, y_M, tol);
```

注意，本函数不允许返回被积函数调用次数，故用户可以自己在被积函数中设置一个计数器，从而测出调用次数。

【例 3.23】 试求出下面的双重定积分

$$J = \int_{-1}^{1} \int_{-2}^{2} \mathrm{e}^{-x^2/2} \sin(x^2+y)\mathrm{d}x\mathrm{d}y$$

由给出的被积函数可以简单地写出下面的函数。

```
function z=my2dfun(x,y)
global kk; kk=kk+1;
z=exp(-x.^2/2).*sin(x.^2+y);
```

这里 kk 为全局变量，每次运行此函数时该值增 1，所以可以用该值作为被积函数计数器。可以通过下面的 MATLAB 语句求出被积函数的双重定积分。

```
>> global kk; kk=0;
   y=dblquad('my2dfun',-2,2,-1,1), kk
```

```
y =
    1.57456866245358
kk =
       1781
```

如果使用 inline() 函数，则上面的命令可以简化为

```
>> f=inline('exp(-x.^2/2).*sin(x.^2+y)','x','y');
   y=dblquad(f,-2,2,-1,1),
y =
    1.57456866245358
```

遗憾的是，在 MATLAB 中并没有提供求解更一般的双重积分问题的函数

$$I=\int_{y_m}^{y_M}\int_{x_m(y)}^{x_M(y)} f(x,y)\mathrm{d}x\mathrm{d}y \tag{3.61}$$

好在美国学者 Howard Wilson 与 Bryce Gardner 开发了数值积分工具箱 (NIT，即 Numerical Integration Toolbox)，该工具箱可以在 The MathWorks 公司的网站上免费下载[1]，该工具箱中函数 gquad2dggen() 可以直接求解式 (3.61) 的双重积分问题，该函数的调用格式为：

J=quad2dggen(被积函数名，下限函数名，上限函数名，y_m，y_M，tol)

其中 tol 为误差限。误差限越小，计算应该越精确，但计算量也将增大，此函数默认的误差限为 tol=10^{-3}。该函数还涉及 3 个 MATLAB 函数，即被积函数和上下限函数。此函数并不能返回被积函数调用次数。下面将通过一个具体例子来演示双重积分的运算。

【例 3.24】 试求出下面的双重定积分

$$J=\int_{-1}^{1}\int_{-\sqrt{1-y^2}}^{\sqrt{1-y^2}} \mathrm{e}^{-x^2/2}\sin(x^2+y)\mathrm{d}x\mathrm{d}y$$

可以构造出 $x_M(y)$ 和 $x_m(y)$ 函数如下：

```
function xh=g_upper(y)
xh=sqrt(1-y.^2);

function xl=g_lower(y)
xl=-sqrt(1-y.^2);
```

这样，可以通过下面的 MATLAB 语句求出被积函数的双重定积分。

```
>> global kk; kk=0;
   y=quad2dggen('my2dfun','g_lower','g_upper',-1,1), kk
y =
```

[1] 可以从作者维护的 "MATLAB 大观园" 中下载。目前该软件是在 MATLAB 4.2 下调试的，在 6.x 版本下可能会有不兼容之处，但数值计算部分的内核不会有问题。另外，安装完该工具箱后，别忘了将其路径加载到 MATLAB 路径下。

```
        0.53685960348552
kk =
     30
>> f=inline('exp(-x.^2/2).*sin(x.^2+y)','x','y');
   fm=inline('-sqrt(1-y.^2)','y'); fM=inline('sqrt(1-y.^2)','y');
   y=quad2dggen(f,fm,fM,-1,1,1e-6),
y =
   0.53686038184820
```

按照例 3.23 改写上下限函数 g_lower() 和 g_upper()，再调用 quad2dggen() 函数，则可以得出如下结果:

```
>> fm=inline('-2','y'); fM=inline('2','y');
   y=quad2dggen(@my2dfun,fm,fM,-1,1,1e-6),
y =
   1.57449815921736
```

可见所需被积函数调用次数明显减少了。再考虑该工具箱建议的求解矩形区域积分的专用函数 quad2dg()，其调用方法类似于 dblquad() 函数，则

```
>> global kk, kk=0; y1=quad2dg(@my2dfun,-2,2,-1,1), kk
y1 =
    1.57449815921723
kk =
     4
```

使用该工具箱中推荐的 quad2dg() 函数，只需调用被积函数 4 次就求出了结果。本矩形边界的双重积分在 Mathematica 下可以求出的精确解为 1.574498159217360522742084，可见即使这里被积函数调用的次数少得多，其精度却大大地高于 MATLAB 本身提供的 dblquad() 函数。

NIT 工具箱还可以解决多重超维长方体边界的定积分问题，如使用 **quadndg()** 函数，另外，其单重积分函数 **quadg()** 的调用格式和 **quad8()** 一致，其效率也高于 **quad8()**，故在进行数值求积分时建议使用此工具箱。

3.3.4 微积分问题的解析解运算

符号运算工具箱中提供了高等数学中的极限、微分、积分和 Taylor 级数展开各种问题的直接求解方法。例如极限可以由 limit() 函数求得，微分和不定积分可以由 diff() 和 int() 函数求出，而 Taylor 幂级数展开可以由 taylor() 函数求出。这些函数的调用都是十分简单、直观的。这里将通过下面的例子来演示这些函数的应用。

【例 3.25】 试求出下面的极限

$$\lim_{x\to\infty}\left(\frac{3x^2-x+1}{2x^2+x+1}\right)^{x^3/(1-x)}$$

若直接给出如下的命令

```
>> syms x, limit(((3*x^2-x+1)/(2*x^2+x+1))^(x^3/(1-x)))
ans =
   1
```

则相当于求 $x \to 0$ 的极限。若想求出 $x \to \infty$ 时的极限，则应该给出如下的命令

```
>> limit(((3*x^2-x+1)/(2*x^2+x+1))^(x^3/(1-x)),x,inf)
ans =
   0
```

〖例 3.26〗 给定一个函数

$$y(x) = \frac{1}{x^2+4x+3}\sin(x)$$

可以由下面的命令对它进行微分运算

```
>> syms x; y='1/(x^2+4*x+3)*sin(x)'; y1= diff(y,x)
y1 =
   -1/(x^2+4*x+3)^2*sin(x)*(2*x+4)+1/(x^2+4*x+3)*cos(x)
>> latex(y1)
ans =
   -{\frac {\sin(x)\left (2\,x+4\right )}{\left({x}^{2}+4\,x
   +3\right )^{2}}}+{\frac {\cos(x)}{{x}^{2}+4\,x+3}}
```

在 LaTeX 系统中该函数将显示成

$$-\frac{\sin(x)\,(2\,x+4)}{(x^2+4\,x+3)^2}+\frac{\cos(x)}{x^2+4\,x+3}$$

另外，可以用 ccode() 函数将之变换成 C 语言表达式

```
>> ccode(y1)
ans =
   t0 = -1/(pow(x*x+4.0*x+3.0,2.0))*sin(x)*(2.0*x+4.0)+1/
        (x*x+4.0*x+3.0)*cos(x);
```

对得出的微分结果再进行积分运算，得出了另一个表达式，经过化简处理，则得出和原函数一致的结果。

```
>> y2=int(y1,x)
y2 =
   -1/2*sin(x)/(x+3)+1/2*sin(x)/(x+1)
>> y2=simple(y2)
y2 =
   sin(x)/(x^2+4*x+3)
>> syms a b; y3=int(y1,x,a,b)
y3 =
   (4*sin(b)*a+sin(b)*a^2+3*sin(b)-4*sin(a)*b-sin(a)*b^2-
   3*sin(a))/(b+3)/(1+b)/(a+3)/(1+a)
```

下面的语句将给定函数 $y(x)$ 对 x 自变量进行 Taylor 级数展开的前 20 项。

```
>> y4=taylor(y,x,20)
y4 =
  1/3*x-4/9*x^2+23/54*x^3-34/81*x^4+4087/9720*x^5-3067/7290*x^6+
  515273/1224720*x^7-386459/918540*x^8+37100281/88179840*x^9-
  5565053/13226976*x^10+36729372941/87298041600*x^11-
  27547035491/65473531200*x^12+17189351351903/40855483468800*x^13-
  1172001255937/2785601145600*x^14+
  10829291689237649/25738954585344000*x^15-
  8121968788023179/19304215939008000*x^16+
  26510106147059012473/63008960824922112000*x^17-
  19882579616032098409/47256720618691584000*x^18+
  533320959163693432453/1267592035419021312000*x^19
```

无穷级数的求和问题可以用 symsum() 函数来求解，该函数的调用格式为：

symsym(通项，起始变量，终止变量)

〖例 3.27〗 考虑下面无穷级数的求解

$$I=2\sum_{n=1}^{\infty}\left(\frac{1}{2^n}+\frac{1}{3^n}\right)$$

可以由下面的 MATLAB 语句求出无穷级数的解。

```
>> syms n; symsum(2/2^n+2/3^n,n,1,inf)
ans =
   3
```

再考虑一个更复杂的问题

$$I=2\sum_{n=0}^{\infty}\frac{1}{(2n+1)(2x+1)^{2n+1}}$$

可以用下面的命令得出该无穷级数的和

```
>> syms n x
   s1=symsum(2/((2*n+1)*(2*x+1)^(2*n+1)),n,0,inf)
ans =
   1/(2*x+1)*(4*x^2+4*x+1)^(1/2)*log((1+1/(4*x^2+4*x+1)^(1/2))
     /(1-1/(4*x^2+4*x+1)^(1/2)))
>> s1=simple(s1)
s1 =
   log((((2*x+1)^2)^(1/2)+1)/(((2*x+1)^2)^(1/2)-1))
```

从得出的结果可以看出，其中包含了某平方数的开方，所以这样的结果不是最简。如果将 x 设置为非零变量，则可以将其化简到最简形式

```
>> syms x nonzero
   simple(s1)
ans =
   log((x+1)/x)
```

其结果和理论值完全一致。由这个例子可以看出，syms 命令不但可以指定某些变量为符号变量，还可以指定其类型，这里的 nonzero 表示非零，此外还可以声明为 real, unreal, positive, negative 等。

【例 3.28】 考虑例 3.24 中给出的双重非矩形区域的定积分问题，可以采用下面的 MATLAB 语句求解出所需的积分值

```
>> syms x y   % 声明符号变量
   f1=exp(-x^2/2)*sin(x^2+y); % 定义被积函数 f1
   f2=int(f1,y,-sqrt(1-x^2),sqrt(1-x^2))   % 求内积分
f2 =
   2*exp(-1/2*x^2)*sin(x^2)*sin((1-x^2)^(1/2))
>> f3=int(f2,-1,1); vpa(f3) % 求定积分并用 vpa 函数取值
Warning: Explicit integral could not be found.
> In c:\matlab6p1\toolbox\symbolic\@sym\int.m at line 58
ans =
    .53686038269880787557759384929130
```

该语句能求解出积分值，但同时指出原来问题的解析解不存在、得到的只是近似解。

再考虑矩形区域的积分，该积分对应的 x 区域为 $[-2,2]$，y 区域为 $[-1,1]$，所以可以使用下面的命令求出高精度的数值解

```
>> f2=int(f1,x,-2,2); % 先求解 y 积分计算速度更快
   vpa(int(f2,y,-1,1))
ans =
   1.5744981592173605227420845222700
```

3.4　常微分方程的数值解法

微分方程初值问题的数值解法实际上是动态系统数字仿真的基础。假设一阶常微分方程组有下式给出

$$\dot{x}_i = f_i(t, \boldsymbol{x}),\ \ i = 1, 2, \cdots, n \tag{3.62}$$

其中 $\boldsymbol{x}$ 为状态变量 x_i 构成的向量，即 $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]^{\mathrm{T}}$，常称为系统的状态向量，$n$ 称为系统的阶次，而 $f_i(\cdot)$ 为任意非线性函数，t 为时间变量，这样就可以采用数值方法在初值 $\boldsymbol{x}(0)$ 下来求解常微分方程组了。

求解常微分方程组的数值方法是多种多样的，如常用的 Euler 法、Runge-Kutta 方法、Adams 线性多步法、Gear 法等。为解决刚性 (stiff) 问题又有若干专用的刚性问题求

解算法，另外，如需要求解隐式常微分方程组和含有代数约束的微分代数方程组时，则需要对方程进行相应的变换，方能进行求解。本节中将给出这些特殊问题的求解方法。

3.4.1 常用常微分方程的数值解法

3.4.1.1 Euler 算法

本节首先介绍最简单的Euler算法。为简单起见，可以将常微分方程组用向量形式表示为

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(t, \boldsymbol{x}) \tag{3.63}$$

式中，$\boldsymbol{f}(\cdot) = [f_1(\cdot), f_2(\cdot), \cdots, f_n(\cdot)]^{\mathrm{T}}$。可以假设，在 t_0 时刻系统状态向量的值为 $\boldsymbol{x}_0$，若选择计算步长 h，则可以写出在 t_0+h 时刻系统状态向量的值为

$$\boldsymbol{x}(t_0+h) = \hat{\boldsymbol{x}}(t_0+h) + \boldsymbol{R}_0 = \boldsymbol{x}_0 + h\boldsymbol{f}(t, \boldsymbol{x}_0) + \boldsymbol{R}_0 \tag{3.64}$$

简记 $\boldsymbol{x}_1 = \boldsymbol{x}(t_0+h)$，则 $\hat{\boldsymbol{x}}_1 = \hat{\boldsymbol{x}}(t_0+h)$ 为系统状态向量在 t_0+h 时刻的近似值，亦即数值解。可见 $\boldsymbol{R}_0$ 为数值解的舍入误差。在实际解法中为简单起见，经常可以舍弃 $\hat{}$ 记号，而将数值解直接记为 $\boldsymbol{x}_1$。

更一般地，假设已知在 t_k 时刻系统的状态向量为 $\boldsymbol{x}_k$，则在 t_k+h 时刻的数值解可以写成

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\boldsymbol{f}(t, \boldsymbol{x}_k) \tag{3.65}$$

这样，用迭代的方法可以由给定的初值问题逐步求出在所选择的时间段 $t \in [0, T]$ 内各个时刻 $t_0+h, t_0+2h, \cdots$ 处的原问题数值解。

提高数值解精度的一种显然的方法是减小步长 h 的值。然而，并不能无限制地减小 h 的值，这主要有两条原因：

(1) **减慢计算速度** 因为对选定的求解时间而言，减小步长就意味着增加在这个时间段内的计算点数目，故计算速度减慢；

(2) **增加累积误差** 因为不论选择多小的步长，所得出的数值解都将有一个舍入误差，减小计算步长则将增加计算的次数，从而使得整个计算过程的舍入误差的叠加和传递次数增多，产生较大的累积误差。舍入误差、累积误差和总误差关系的示意图如图 3-2 所示。

所以在对动态系统进行仿真分析时，应采取下列措施：

(1) **选择适当的步长** 采用像 Euler 法这样简单的算法时，应适当地选择步长，既不能太大，又不能太小；

(2) **改进近似算法精度** 由于 Euler 算法只是将原积分问题进行梯形的近似，其近似精度很低，从而不能很有效地逼近原始问题。可以用各种更精确的插值方法来取代 Euler 公式，从而改进运算精度。比较成功的是 Runge-Kutta 法、Adams 法等；

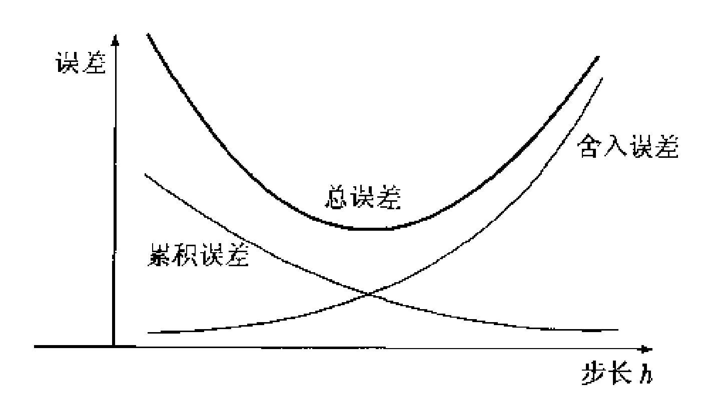


图 3-2 各种误差示意图

(3) **采用变步长方法** 前面提及“适当”地选择步长，这本身就是个模糊的概念，如何适当地选择步长取决于经验。事实上，很多种方法都允许变步长的求解，如果误差较小时，可自动地增大步长，而误差较大时再自动减小步长，从而精确、有效地求解给出的常微分方程初值问题。

3.4.1.2 Runge-Kutta 算法

- **二阶** Runge-Kutta **算法**

记在 t_k 时刻的状态向量为 $\boldsymbol{x}_k$，并定义两个附加向量型变量

$$\begin{cases} \boldsymbol{K}_1 = h\boldsymbol{f}(t_k, \boldsymbol{x}_k) \\ \boldsymbol{K}_2 = h\boldsymbol{f}(t_k + h, \boldsymbol{x}_k + \boldsymbol{K}_1) \end{cases} \tag{3.66}$$

则二阶 Runge-Kutta 法数值解可以由下式得出

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{1}{2}(\boldsymbol{K}_1 + \boldsymbol{K}_2) \tag{3.67}$$

- **四阶** Runge-Kutta **算法**

定义四个附加向量

$$\begin{cases} \boldsymbol{K}_1 = h\boldsymbol{f}(t_k, \boldsymbol{x}_k) \\ \boldsymbol{K}_2 = h\boldsymbol{f}(t_k + h/2, \boldsymbol{x}_k + \boldsymbol{K}_1/2) \\ \boldsymbol{K}_3 = h\boldsymbol{f}(t_k + h/2, \boldsymbol{x}_k + \boldsymbol{K}_2/2) \\ \boldsymbol{K}_4 = h\boldsymbol{f}(t_k + h, \boldsymbol{x}_k + \boldsymbol{K}_3) \end{cases} \tag{3.68}$$

则四阶 Runge-Kutta 法数值解可以由下式得出

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{1}{6}(\boldsymbol{K}_1 + 2\boldsymbol{K}_2 + 2\boldsymbol{K}_3 + \boldsymbol{K}_4) \tag{3.69}$$

这样，用迭代的方法由给定的初值问题逐步求出在所选择的时间段 $t \in [0,T]$ 内各个时刻 $t_0+h, t_0+2h, \cdots$ 处的原问题数值解。

有了上面的数学算法，则可以用 MATLAB 语言容易地编写出该算法的函数

```
function [tout,yout] = rk_4(odefile,tspan,y0)
t0=tspan(1); th=tspan(2);
if length(tspan)<=3, h=tspan(3);
else, h=tspan(2)-tspan(1); th=tspan(2); end
tout=[t0:h:th]'; yout=[];
for t=tout'
   k1=h*eval([odefile '(t,y0)']);
   k2=h*eval([odefile '(t+h/2,y0+0.5*k1)']);
   k3=h*eval([odefile '(t+h/2,y0+0.5*k2)']);
   k4=h*eval([odefile '(t+h,y0+k3)']);
   y0=y0+(k1+2*k2+2*k3+k4)/6;
   yout=[yout; y0'];
end
```

其中，tspan 可以有两种构成方法：第一种方法可以是一个等间距的时间向量；第二种方法是 tspan=[t0, th, h]，其中 t0 和 th 为计算的初始和终止值，h 为计算步长，odefile 是一个字符串变量，为表示微分方程的文件名，y0 是初值列向量。函数调用完成后，时间向量与各个时刻状态变量构成的矩阵分别由 tout 和 yout 返回。

- **四阶五级 Runge-Kutta 变步长算法**

德国学者 Felhberg 对传统的 Runge-Kutta 方法进行了改进[9]，在每一个计算步长内对 $f_i(\cdot)$ 函数进行六次求值，以保证更高的精度和数值稳定性，该算法又称为四阶五级 FRK 算法。假设当前的步长为 h_k，则可以定义下面的 6 个 $\boldsymbol{K}_i$ 变量

$$\boldsymbol{K}_i = h_k \boldsymbol{f}\left(t_k+\alpha_i h_k, \boldsymbol{x}_k+\sum_{j=1}^{i-1}\beta_{ij}\boldsymbol{K}_j\right),\ \ i=1,2,\cdots,6 \tag{3.70}$$

式中 t_k 为当前计算时刻，而中间参数 α_i, β_{ij} 及其他参数由表 3-3 给出，其中 α_i, β_{ij} 参数对又称为 Dormand-Prince 对。这时下一步的状态向量可以由下式求出

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \sum_{i=1}^{6}\gamma_i \boldsymbol{K}_i \tag{3.71}$$

当然直接采用这一方法则为定步长的方法。在实际问题中往往希望在一些情况下 (如解变化很快时) 采用较小的步长，而在另一些情况下 (如解的变化很缓慢时) 采用较大的步长，这样做既可以保证较高的精度，又可以保证较高的运算速度。在此算法中，定义一个误差向量 $\epsilon_k = \sum_{i=1}^{6}(\gamma_i-\gamma_i^*)\boldsymbol{K}_i$，可以由它的大小来变换计算步长，所以这种能自动变换步长的方法又称为自适应变步长方法。

表 3-3 四阶五级 RKF 算法系数表

α_i	β_{ij}					γ_i	γ_i^*
0						16/135	25/216
1/4	1/4					0	0
3/8	3/32	9/32				6656/12825	1408/2565
12/13	1932/2197	-7200/2197	7296/2197			28561/56430	2197/4104
1	439/216	-8	3680/513	-845/4104		-9/50	-1/5
1/2	-8/27	2	-3544/2565	1859/4104	-11/40	2/55	0

3.4.1.3 Adams 算法

- **Adams 预报公式**　Adams 预报公式又称为外推公式：

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\frac{h}{24}\Big[55\boldsymbol{f}(t_k,\boldsymbol{x}_k)-59\boldsymbol{f}(t_{k-1},\boldsymbol{x}_{k-1})+37\boldsymbol{f}(t_{k-2},\boldsymbol{x}_{k-2})-9\boldsymbol{f}(t_{k-3},\boldsymbol{x}_{k-3})\Big] \tag{3.72}$$

其中 $k=3,4,\cdots$。可以看出，只给定 $\boldsymbol{x}_0$ 向量初值是不能采用本算法直接求解的，因为该公式还需要已知 $\boldsymbol{x}_1,\boldsymbol{x}_2$ 和 $\boldsymbol{x}_3$ 的值，而这些值可以由其他的算法，如 Runge-Kutta 法求出，然后再采用 Adams 外推公式即可。所以，Adams 算法是不能自动启动的，它要求其他算法提供若干点的初值。

- **Adams 校正公式**　Adams 校正公式又称为内插公式：

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\frac{h}{24}\Big[9\boldsymbol{f}(t_{k+1},\boldsymbol{x}_{k+1})+19\boldsymbol{f}(t_k,\boldsymbol{x}_k)-5\boldsymbol{f}(t_{k-1},\boldsymbol{x}_{k-1})+\boldsymbol{f}(t_{k-2},\boldsymbol{x}_{k-2})\Big] \tag{3.73}$$

其中 $k=2,3,\cdots$。可以看出，这样得出的公式是关于 $\boldsymbol{x}_{k+1}$ 的隐式公式，所以在等号右面的 $\boldsymbol{x}_{k+1}$ 项又常常使用 Adams 预报公式得出的值，这样改进的算法又称为 Adams 预报校正公式。同样，前几个 $\boldsymbol{x}_k$ 的值仍需由其他算法求出。

3.4.1.4 Gear 算法

类似于四阶 Runge-Kutta 算法，在 Gear 算法中需要构造如下的中间向量

$$\begin{cases}\boldsymbol{K}_1=h\boldsymbol{f}(t_k,\boldsymbol{x}_k)\\ \boldsymbol{K}_2=h\boldsymbol{f}\left(t_k+\dfrac{h}{2},x_k+\dfrac{\boldsymbol{K}_1}{2}\right)\\ \boldsymbol{K}_3=h\boldsymbol{f}\left(t_k+\dfrac{h}{2},x_k+\left(\dfrac{-1}{2}+\sqrt{\dfrac{1}{2}}\right)\boldsymbol{K}_1+\left(1-\sqrt{\dfrac{1}{2}}\right)\boldsymbol{K}_2\right)\\ \boldsymbol{K}_4=h\boldsymbol{f}\left(t_k+h,\boldsymbol{x}_k-\sqrt{\dfrac{1}{2}}\boldsymbol{K}_2+\left(1+\sqrt{\dfrac{1}{2}}\right)\boldsymbol{K}_3\right)\end{cases} \tag{3.74}$$

则 Gear 法数值解可以由下式得出

$$\boldsymbol{x}_{k+1}=\boldsymbol{x}_k+\frac{1}{6}\boldsymbol{K}_1+\frac{1}{3}\left(1-\sqrt{\frac{1}{2}}\right)\boldsymbol{K}_2+\frac{1}{3}\left(1+\sqrt{\frac{1}{2}}\right)\boldsymbol{K}_3+\frac{1}{6}\boldsymbol{K}_4 \tag{3.75}$$

3.4.2 MATLAB 下的常微分方程求解函数

MATLAB 提供了两个常微分方程求解的函数 ode23() 和 ode45()。这两个函数分别采用了二阶三级的 RKF 方法和四阶五级的 RKF 方法，并采用自适应变步长的求解方法，即当解的变化较慢时采用较大的计算步长，从而使得计算速度很快，当方程的解变化得较快时，积分步长会自动地变小，从而使得计算的精度很高。这两个函数的调用格式分别为：

```
[t, x]=ode23(方程函数名, tspan, x0, 选项, 附加参数)
[t, x]=ode45(方程函数名, tspan, x0, 选项, 附加参数)
```

其中，"选项"可以通过 odeget() 和 odeset() 函数来设置，具体的常用选项如下：

- RelTol —— 为相对误差容许上限，默认值为 0.001 (即 0.1% 的相对误差)，在一些特殊的微分方程求解中，为了保证较高的精度，还应该再适当减小该值。
- AbsTol —— 为一个向量，其分量表示每个状态变量允许的绝对误差，其默认值为 10^{-6}。当然可以自由设置其值，以改变求解精度。
- MaxStep —— 为求解方程最大允许的步长。
- Mass —— 微分代数方程中的质量函数。
- Jacobian —— 为描述 Jacob 矩阵函数 $\partial \boldsymbol{f}/\partial \boldsymbol{x}$ 的函数名，如果已知该 Jacob 矩阵，则能加速仿真过程。

在一般应用中没有必要修改其默认属性，可以直接采用默认值。这里所用到的方程函数名为描述系统状态方程的 M 函数的名称，该函数名应该用引号括起来。

变量 tspan 一般为仿真范围，例如取 tspan=[t0,tf]，其中 t0 和 tf 分别为用户指定的起始和终止计算时间，值得指出的是，MATLAB 从 5.0 版开始允许 t0 的值小于 tf。另外，tspan 还可以取作不等间距的时间向量。变量 x0 为系统的初始状态变量的值，注意，应该使该向量的元素个数等于系统状态变量的个数，否则将给出错误信息。如果想采用默认的零初始状态，则可以在该处给一个空矩阵。

有了这些参数，就可以调用 ode23() 和 ode45() 两个函数对系统直接进行求解了。此函数将返回两个变量 t 和 x，其中 t 为求解的时间变量，因为采用了变步长的求解算法，所以得出的 t 向量并不一定是等间隔的；另一个变量 x 为状态变量在各个时刻所组成的列向量所构成的矩阵的转置。有了 t 和 x 这样的变量，在求解过程结束后即可以用 plot(t,x) 来绘制出解的结果曲线。

这里要用到的方程函数名的编写格式是固定的，如果其格式没有按照要求去编写，则将得出错误的求解结果。方程函数的引导语句为：

```
function xdot= 方程函数名(t,x,flag,附加参数)
```

其中 t 为时间变量，x 为方程的状态变量，而 xdot 为状态变量的导数。注意，即使微分方程是非时变的，也应该在函数输入变量列表中写上 t 占位。可见，如果想编写这样的函数，首先必须已知原系统的状态方程模型。

如果有附加参数需要传递，则可以将其在原函数中给出，若有多个附加参数，则它们之间应该用逗号分隔，且应确保它们与主调函数完全对应。另外应该用一个变量 **flag** 来占位。

3.4.3 常微分方程举例

在本节中通过几个例子来演示 MATLAB 中常微分方程求解的方法，并指出求解过程中可能遇到的问题和这些问题的解决方案。

〖**例** 3.29〗 假设著名的 Lorenz 模型的状态方程表示为

$$\begin{cases} \dot{x}_1(t) = -8x_1(t)/3 + x_2(t)x_3(t) \\ \dot{x}_2(t) = -10x_2(t) + 10x_3(t) \\ \dot{x}_3(t) = -x_1(t)x_2(t) + 28x_2(t) - x_3(t) \end{cases}$$

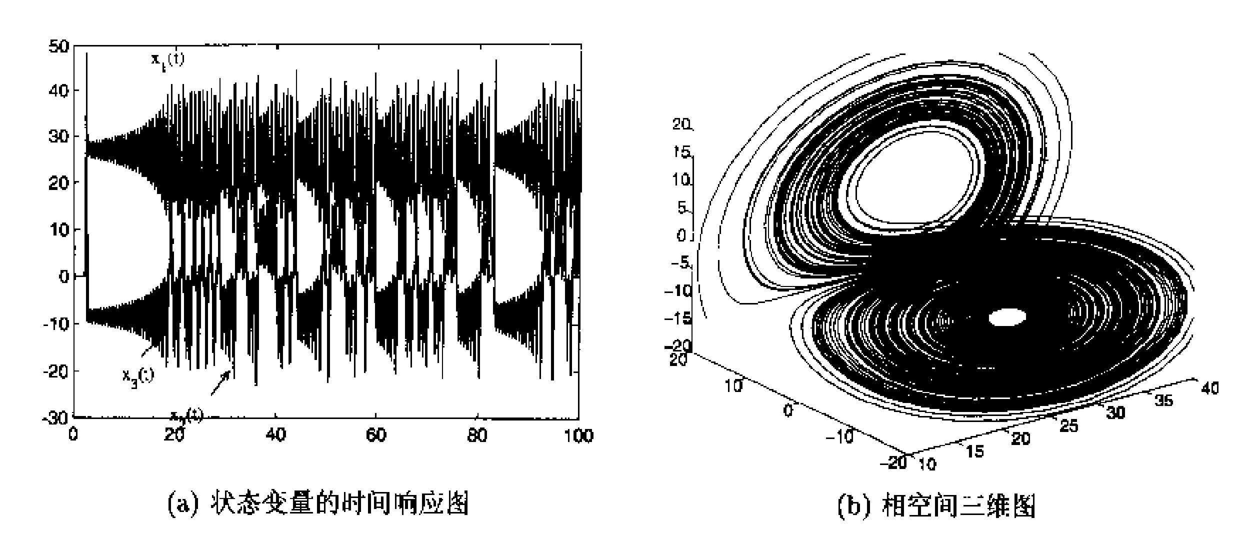
若令其初值为 $x_1(0) = x_2(0) = 0$, $x_3(0) = \epsilon$，而 ϵ 为机器上可以识别的小常数，例如取 $\epsilon = 10^{-10}$，则可以按下面的格式编写出一个 MATLAB 函数 lorenzeq.m 来描述系统的动态模型，其内容为：

```
function xdot = lorenzeq(t,x)
xdot=[-8/3*x(1)+x(2)*x(3);
      -10*x(2)+10*x(3);
      -x(1)*x(2)+28*x(2)-x(3)];
```

这时，可以调用微分方程数值解 ode45() 函数对 lorenzeq() 函数描述的系统进行数值求解，并将结果进行图形显示。

```
>> t_final=100; x0=[0;0;1e-10];
   [t,x]=ode45('lorenzeq',[0,t_final],x0);
   plot(t,x),
   figure; plot3(x(:,1),x(:,2),x(:,3)); axis([10 40 -20 20 -20 20]);
```

其中，t_final 为设定的仿真终止时间，x0 为初始状态。第一个绘图命令绘制出系统的各个状态和时间关系的二维曲线图，如图 3-3(a) 所示。第二个绘图命令可以绘制出三个状态的相空间曲线，如图 3-3(b) 所示。可以看出，看似很复杂的三元一阶常微分方程组的数值解问题由几条简单直观的 MATLAB 语句就求解出来了，此外，用 MATLAB 语言还可以轻易、直观地将结果直接显示出来，这就是我们将 MATLAB 语言作为本书的主要语言的原因。



(a) 状态变量的时间响应图　　(b) 相空间三维图

图 3-3 Lorenz 方程的仿真结果图示

〖例 3.30〗 考虑著名的 Van der Pol 方程

$$\ddot{y}+\mu(y^2-1)\dot{y}+y=0$$

选择状态变量 $x_1=y, x_2=\dot{y}$，则原方程可以变换成

$$\dot{x}_1=x_2,\ \dot{x}_2=-\mu(x_1^2-1)x_2-x_1$$

这里的 μ 是一个可变参数，如果对每一个要研究的 μ 值都编写一个函数则显得不方便，所以应该采用附加参数的概念将 μ 的值传给该函数，这样可以如下写出描述此模型的 M 函数为

```
function y=vdp_eq(t,x,flag,mu)
y=[x(2);
   -mu*(x(1).^2-1).*x(2)-x(1)];
```

可见，在函数定义时多了一个 mu 项，该项应该在 ode45() 函数调用时传给 vdp_eq() 函数。可以在 MATLAB 工作空间中给 mu 赋值，再进行方程求解。

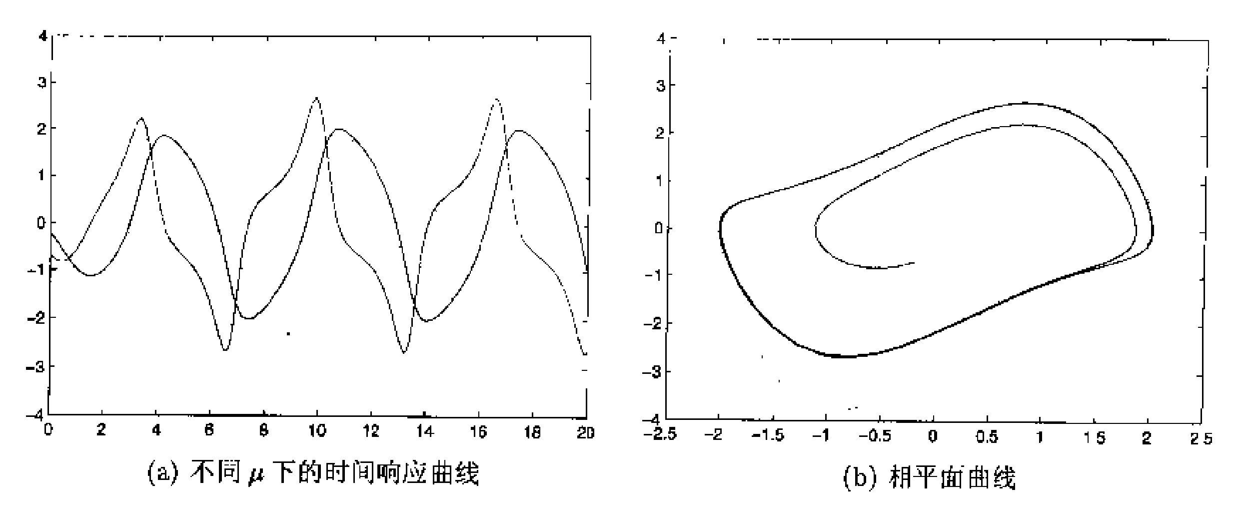
假定初值为 $x=[-0.2;-0.7]$，则最终的求解函数格式为

```
>> h_opt=odeset; x0=[-0.2; -0.7]; t_final=20;
   mu=1; [t1,y1]=ode45('vdp_eq',[0,t_final],x0,h_opt,mu);
   mu=2; [t2,y2]=ode45('vdp_eq',[0,t_final],x0,h_opt,mu);
   plot(t1,y1,t2,y2,':')
   figure; plot(y1(:,1),y1(:,2),y2(:,1),y2(:,2),':')
```

这样在 $\mu=1,2$ 时的时间响应曲线和相平面曲线如图 3-4 所示。

注意，在定义函数时应该有个 flag 变量，其作用是用来指定初值的。即使初值不用指定，也必须有该变量占位。调用函数 ode45() 时也应该给出选项变量占位。在 ode45() 调用命令中的附加变量个数应该和方程 M 函数中的附加参数个数完全对应，否则将出现错误结果。

使用“函数句柄”的概念、其函数编写的格式与前面介绍的一致，所不同的是，实际应用中



(a) 不同 μ 下的时间响应曲线

(b) 相平面曲线

图 3-4 不同 μ 值下 Van der Pol 方程解

若需要带附加参数，则在写函数文件时不用 flag 变量占位，这样在 ode45() 时，则不用引用函数名，而直接用句柄即可。

采用函数句柄的概念，则可以将描述系统的函数改写为

```
function y=vdp_eq1(t,x,mu)
y=[x(2);
   -mu*(x(1).^2-1).*x(2)-x(1)];
```

这时可以认为 vdp_eq1 为函数句柄，这样就可以用下面的命令调用该函数句柄，求解原始方程，最终得出同样的结果。

```
>> h_opt=odeset; x0=[-0.2; -0.7]; t_final=20;
   mu=1; [t1,y1]=ode45(@vdp_eq1,[0,t_final],x0,h_opt,mu);
   mu=2; [t2,y2]=ode45(@vdp_eq1,[0,t_final],x0,h_opt,mu);
   plot(y1(:,1),y1(:,2),y2(:,1),y2(:,2),':')
```

改变 μ 的值，令 $\mu = 1000$，并设仿真终止时间为 3000，则可以采用下面的命令求解相应的 Van der Pol 方程

```
>> h_opt=odeset; x0=[2;0]; t_final=3000;
   mu=1000; [t,y]=ode45(@vdp_eq1,[0,t_final],x0,h_opt,mu);
```

经过长时间的等待，则发现得出下面的错误信息：

```
??? Error using ==> vertcat
Out of memory. Type HELP MEMORY for your options.
Error in ==> c:\matlab6p1\toolbox\matlab\funfun\ode45.m
On line 392  ==>          yout = [yout; zeros(chunk,neq)];
```

事实上由于变步长所采用的步长过小，而要求的仿真终止时间比较大，导致输出的 y 矩阵过大，超出了计算机存储空间的容限。所以，这个问题不适合采用 ode45() 来求解，后面将采用刚性方程求解的算法来解决这个问题。

〖例 3.31〗 假设给定的隐式微分方程如下

$$\begin{cases} \sin x_1 \dot{x}_1 + \cos x_2 \dot{x}_2 + x_1 = 1 \\ -\cos x_2 \dot{x}_1 + \sin x_1 \dot{x}_2 + x_2 = 0 \end{cases}$$

令 $\boldsymbol{x} = [x_1, x_2]^T$，则可以将原方程改写成矩阵形式：$\boldsymbol{A}(\boldsymbol{x})\dot{\boldsymbol{x}} = \boldsymbol{B}(\boldsymbol{x})$，其中

$$\boldsymbol{A}(\boldsymbol{x}) = \begin{bmatrix} \sin x_1 & \cos x_2 \\ -\cos x_2 & \sin x_1 \end{bmatrix},\quad \boldsymbol{B}(\boldsymbol{x}) = \begin{bmatrix} 1 - x_1 \\ -x_2 \end{bmatrix}$$

如果能证明 $\boldsymbol{A}(\boldsymbol{x})$ 为非奇异的矩阵，则直接就能将该方程变换成标准的一阶微分方程组的形式，即 $\dot{\boldsymbol{x}} = \boldsymbol{A}^{-1}(\boldsymbol{x})\boldsymbol{B}(\boldsymbol{x})$，套用各种 MATLAB 函数求解对应的方程。事实上，由于 MATLAB 能较好地处理奇异问题，所以可以依赖 MATLAB 来求解矩阵的逆，看是否有奇异的错误信息提示，如果没有相应的错误信息则应该相信得出的解。这样可以由下面的函数描述该方程：

```
function dx=c3ximp(t,x)
A=[sin(x(1)) cos(x(2)); -cos(x(2)) sin(x(1))];
B=[1-x(1); -x(2)]; dx=inv(A)*B;
```

可以给出下面的命令求解方程

```
>> [t,x]=ode45(@c3ximp,[0,10],[0; 0]); plot(t,x)
```

同时将获得状态变量的时间曲线，如图 3-5 所示。在求解的过程中也没有得出有关矩阵奇异的错误信息，故得出的结果是可信的。

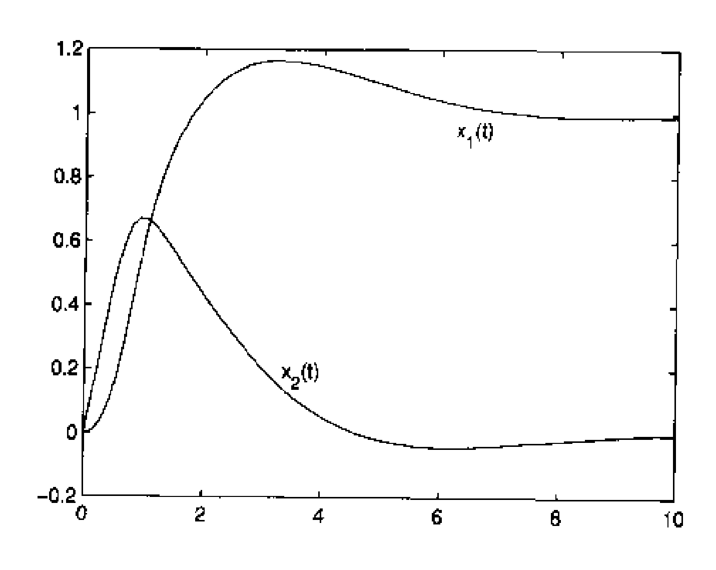


图 3-5 隐式方程的时间响应曲线

3.4.4 刚性方程的 MATLAB 求解

在许多领域中，经常遇到一类特殊的常微分方程，其中一些解变化缓慢，另一些变化快，且相差较悬殊，这类方程常常称为刚性方程，又称为 Stiff 方程。刚性问题一般不适合由 ode45() 这类函数求解，而应该采用 MATLAB 求解函数 ode15s()，该函数调用

格式为：

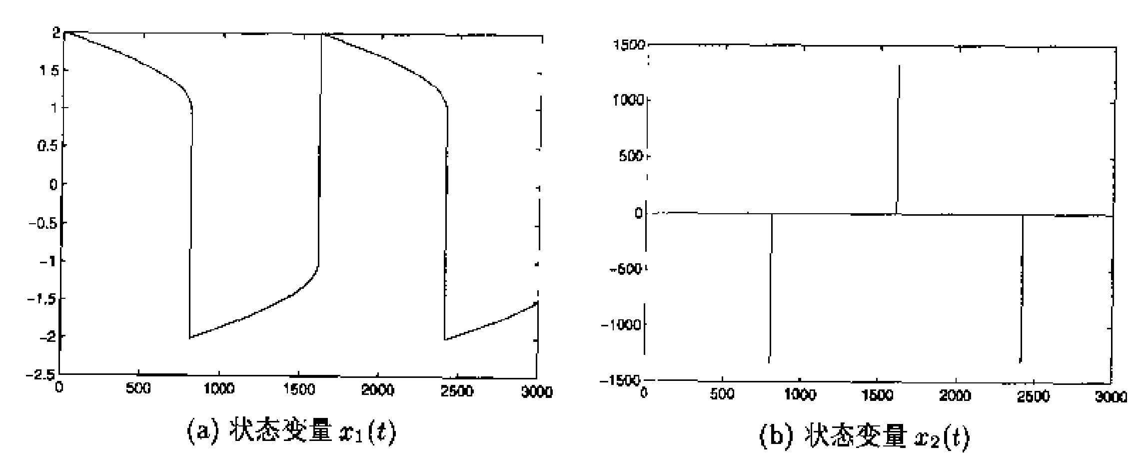
```
[t, x]=ode15s(方程函数名, tspan, x0, 选项, 附加参数)
```

可见，此函数的调用格式和 ode45() 等完全一致。

〖例 3.32〗 首先重新研究前面 $\mu=1000$ 时的 Van der Pol 方程求解问题，仿照上节可以给出如下的 MATLAB 命令

```
>> h_opt=odeset; x0=[2;0]; t_final=3000;
   tic, mu=1000; [t,y]=ode15s(@vdp_eq1,[0,t_final],x0,h_opt,mu);toc
elapsed_time =
     1.3800
>> plot(t,y(:,1)); figure; plot(t,y(:,2))
```

可见，用刚性方程求解函数可以快速求出该方程的数值解，并将两个状态变量的时间曲线分别绘制出来，如图 3-6 所示。从得出的图形可以看出，$x_1(t)$ 曲线变化较平滑，而 $x_2(t)$ 变化在某



(a) 状态变量 $x_1(t)$　　(b) 状态变量 $x_2(t)$

图 3-6　$\mu=1000$ 时 Van der Pol 方程的解

些点上较快。所以当 $\mu=1000$ 时，Van der Pol 方程属于典型的刚性方程，应该采用刚性方程的函数求解。

〖例 3.33〗 在传统的有关常微分方程数值解的教科书[49] 中，都认为下面的微分方程是刚性的

$$\dot{\boldsymbol{y}}=\begin{bmatrix}-21 & 19 & -20\\ 19 & -21 & 20\\ 40 & -40 & -40\end{bmatrix}\boldsymbol{y},\ \ \boldsymbol{y}_0=\begin{bmatrix}1\\0\\-1\end{bmatrix}$$

经过简单的推导，可以得出原问题的解析解为

$$\boldsymbol{y}(t)=\begin{bmatrix}0.5\mathrm{e}^{-2t}+0.5\mathrm{e}^{-40t}(\cos 40t+\sin 40t)\\ 0.5\mathrm{e}^{-2t}-0.5\mathrm{e}^{-40t}(\cos 40t+\sin 40t\\ -\mathrm{e}^{-40t}(\cos 40t-\sin 40t)\end{bmatrix}$$

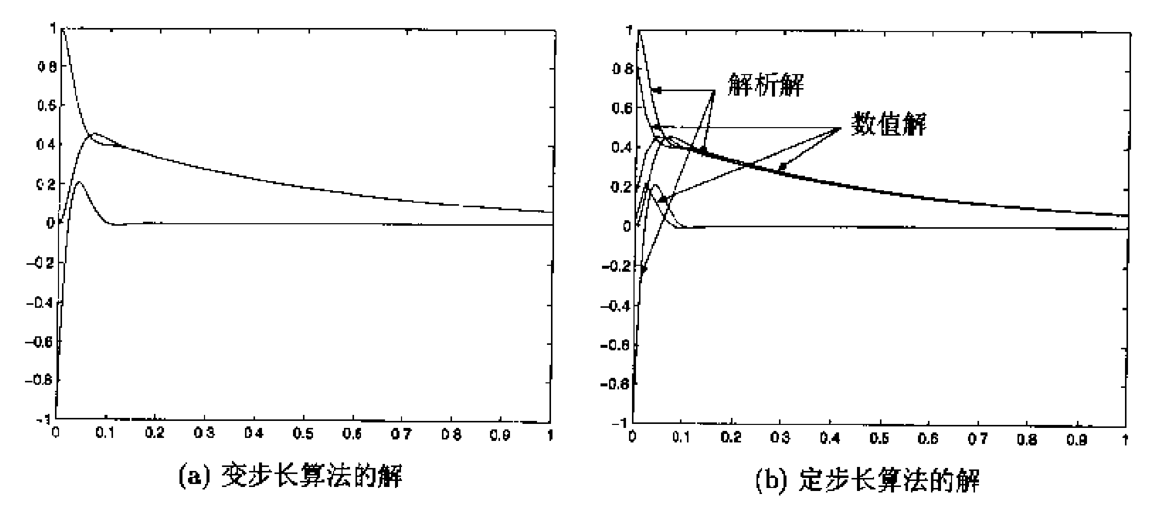
根据原始问题，可以立即写出该模型的 MATLAB 表示

```
function dy=c3xstf1(t,x)
dy=[-21, 19, -20; 19, -21, 20; 40, -40, -40]*x;
```

利用下面的 MATLAB 语句，则可以得出方程的数值解

```
>> tic,[t,y]=ode45(@c3xstf1,[0,1],[1;0;-1]); toc
elapsed_time =
     0.0900
>> x1=exp(-2*t); x2=exp(-40*t).*cos(40*t); x3=exp(-40*t).*sin(40*t);
   y1=[0.5*x1+0.5*x2+0.5*x3, 0.5*x1-0.5*x2-0.5*x3, -x2+x3];
   plot(t,y,t,y1), [max(abs(y-y1)),max(diff(t))]
ans =
    0.0001    0.0001    0.0002    0.0205
```

原方程的解析解和数值解如图 3-7 (a) 所示。可以看出，问题的数值解的精度还是比较高的，计算速度相对也较快，从这里似乎看不出原问题的刚性所在。究其原因，因为在 MATLAB 下采用了变步长的算法，它可以依照要求的精度自动地修正步长，所以感受不到它是个刚性问题。



(a) 变步长算法的解　　(b) 定步长算法的解

图 3-7　给定的传统刚性问题解法比较

如果采用定步长方法，利用前面编写的四阶 Runge-Kutta 定步长算法程序 rk_4()，再用下面的语句就能求解原方程了。

```
>> tic,[t2,y2]=rk_4('c3exstf1',[0,1],[1;0;-1]); toc
elapsed_time =
     0.1610
>> plot(t,y1,t2,y2)
```

这样得出的曲线与解析解的对比见图 3-7 (b)。从计算的结果看，显然采用定步长的算法在取较大的步长时，得出的解是不正确的，同时，采用这样的算法计算速度也较慢，所以在实际应用最好采用变步长算法。

从得出的曲线可以看出，相比之下，这三条曲线的变化速度差别不是特别悬殊，在以前计算能力受限时被误认为是刚性问题，在 MATLAB 下则可以由普通的函数求解。

由此可以得出结论，许多传统的刚性问题采用 MATLAB 的普通求解函数就可以直接解出，而不必刻意地去选择刚性问题的解法。当然在有些问题的求解中确实需要采用刚性问题的解法，将在下个例子中加以说明。

【例 3.34】 考虑下面的常微分方程

$$\begin{cases} \dot{y}_1 = 0.04(1-y_1) - (1-y_2)y_1 + 0.0001(1-y_2)^2 \\ \dot{y}_2 = -10^4 y_1 + 3000(1-y_2)^2 \end{cases}$$

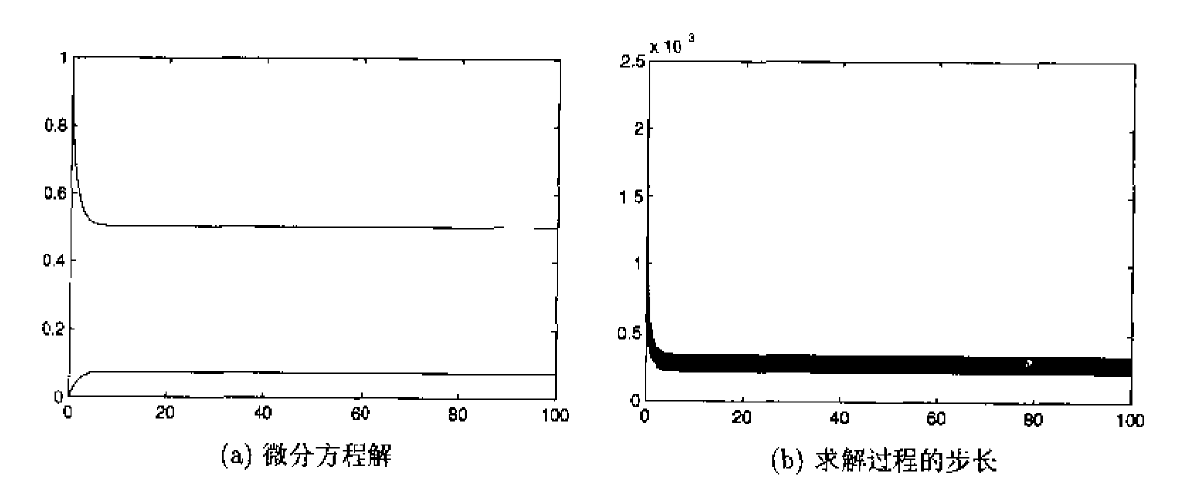
其中，该方程的初值为 $y_1(0)=0, y_2(0)=1$。取计算区间为 $t \in (0,100)$，则根据给出的微分方程，可以写出下面的 MATLAB 函数：

```
function dy=c3exstf2(t,y)
dy=[0.04*(1-y(1))-(1-y(2))*y(1)+0.0001*(1-y(2))^2;
    -10^4*y(1)+3000*(1-y(2))^2];
```

在 MATLAB 的命令窗口中给出下面的语句：

```
>> tic,[t2,y2]=ode45('c3exstf2',[0,100],[0;1]);toc
elapsed_time =
   231.9030
>> length(t2), plot(t2,y2)
ans =
      356941
```

再经过长时间的等待，可以得出原问题的数值解，如图 3-8 (a) 所示。可以看出，调用普通的解法



(a) 微分方程解 (b) 求解过程的步长

图 3-8 四阶五级 Runge-Kutta 法结果

函数 ode45() 则计算所需的时间过长，计算的点也过多，对这个例子来说，计算的点高达 35 万。再分析变步长解法所使用的步长

```
>> format long, [min(diff(t2)), max(diff(t2))]
ans =
   0.00022220693884   0.00214971787184
```

```
>> plot(t2(1:end-1),diff(t2))
```

则可以看出，由于设定的精度要求较高，不得不采用小步长来解决问题，实际的步长如图 3-8(b) 所示，可见在大部分时间内，所采用的步长小于 0.0004，这使得解题时间大大增加。

考虑用 ode15s() 替代 ode45()，则可以得出如下的结果：

```
>> tic,[t1,y1]=ode15s('c3exstf2',[0,100],[0;1]);toc
elapsed_time =
    0.0900
>> length(t1), plot(t1,y1)
ans =
    56
```

可见解题时间大大减小，效率提高了 2500 多倍，得出的曲线则几乎完全一致。

3.4.5 微分方程组的变换与技巧

由前面介绍的微分方程求解函数和微分方程标准型可见，如果常微分方程由一个或多个高阶常微分方程给出，要得出该方程的数值解，则应该先将该方程变换成一阶常微分方程组。这里将分两种情况加以考虑。

3.4.5.1 单个高阶常微分方程处理方法

假设一个高阶常微分方程的一般形式为

$$y^{(n)} = f(t, y, \dot{y}, \cdots, y^{(n-1)}) \tag{3.76}$$

且已知输出变量 $y(t)$ 的各阶导数初始值为 $y(0), \dot{y}(0), \cdots, y^{(n-1)}(0)$，则可以选择一组状态变量 $x_1 = y, x_2 = \dot{y}, \cdots, x_n = y^{(n-1)}$，这样，就可以将原高阶常微分方程模型变换成下面的一阶方程组形式

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_3 \\ \quad\vdots \\ \dot{x}_n = f(t, x_1, x_2, \cdots, x_n) \end{cases} \tag{3.77}$$

且初值 $x_1(0) = y(0), x_2(0) = \dot{y}(0), \cdots, x_n(0) = y^{(n-1)}(0)$。这样变换以后则可以直接求取原方程的数值解了。

3.4.5.2 高阶常微分方程组的变换方法

这里以两个高阶微分方程构成的微分方程组为例介绍如何将之变换成一个一阶常微分方程组。如果可以显式地将两个方程写成

$$\begin{cases} x^{(m)} = f(t, x, \dot{x}, \cdots, x^{(m-1)}, y, \cdots, y^{(n-1)}) \\ y^{(n)} = g(t, x, \dot{x}, \cdots, x^{(m-1)}, y, \cdots, y^{(n-1)}) \end{cases} \tag{3.78}$$

则仍旧可以选择状态变量$x_1 = x, x_2 = \dot{x}, \cdots, x_m = x^{(m-1)}, x_{m+1} = y, x_{m+2} = \dot{y}, \cdots, x_{m+n} = y^{(n-1)}$，这样就可以将原方程变换成

$$\begin{cases} \dot{x}_1 = x_2 \\ \quad\vdots \\ \dot{x}_m = f(t, x_1, x_2, \cdots, x_{m+n}) \\ \dot{x}_{m+1} = x_{m+2} \\ \quad\vdots \\ \dot{x}_{m-n} = g(t, x_1, x_2, \cdots, x_{m+n}) \end{cases} \tag{3.79}$$

再对初值进行相应的变换，就可以得出所期望的一阶微分方程组了。

〖例 3.35〗 已知 Apollo 卫星的运动轨迹 (x, y) 满足下面的方程

$$\ddot{x} = 2\dot{y} + x - \frac{\mu^*(x+\mu)}{r_1^3} - \frac{\mu(x-\mu^*)}{r_2^3},\ \ddot{y} = -2\dot{x} + y - \frac{\mu^* y}{r_1^3} - \frac{\mu y}{r_2^3}$$

其中 $\mu = 1/82.45$, $\mu^* = 1-\mu$, $r_1 = \sqrt{(x+\mu)^2 + y^2}$、$r_2 = \sqrt{(x-\mu^*)^2 + y^2}$，试在初值 $x(0) = 1.2$, $\dot{x}(0) = 0$, $y(0) = 0$, $\dot{y}(0) = -1.04935751$ 下进行求解，并绘制出 Apollo 位置的 (x, y) 轨迹。

可以选择一组状态变量 $x_1 = x, x_2 = \dot{x}, x_3 = y, x_4 = \dot{y}$，这样就可以得出一阶常微分方程组

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = 2x_4 + x_1 - \mu^*(x_1+\mu)/r_1^3 - \mu(x_1 - \mu^*)/r_2^3 \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = -2x_2 + x_3 - \mu^* x_3/r_1^3 - \mu x_3/r_2^3 \end{cases}$$

式中 $r_1 = \sqrt{(x_1+\mu)^2 + x_3^2}$, $r_2 = \sqrt{(x_1-\mu^*)^2 + x_3^2}$，且 $\mu = 1/82.45, \mu^* = 1-\mu$。

有了数学模型描述，则可以立即写出其相应的 MATLAB 函数

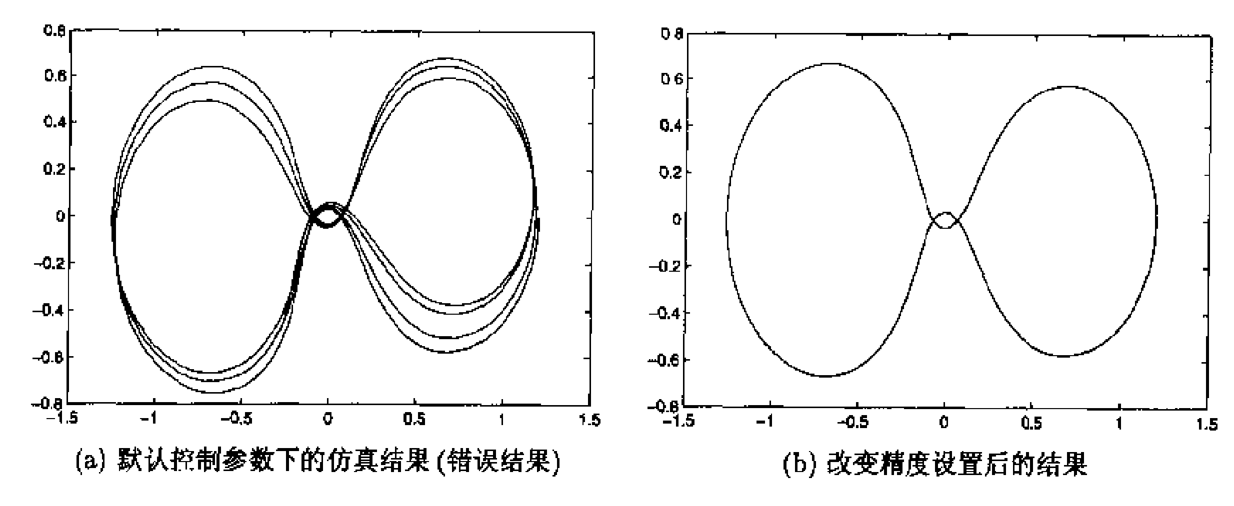
```
function dx=apolloeq(t,x)
mu=1/82.45; mu1=1-mu;
r1=sqrt((x(1)+mu)^2+x(3)^2); r2=sqrt((x(1)-mu1)^2+x(3)^2);
dx=[x(2);
    2*x(4)+x(1)-mu1*(x(1)+mu)/r1^3-mu*(x(1)-mu1)/r2^3;
    x(4);
   -2*x(2)+x(3)-mu1*x(3)/r1^3-mu*x(3)/r2^3];
```

调用 ode45() 函数则可以求出该方程的数值解

```
>> x0=[1.2; 0; 0; -1.04935751];
   tic, [t,y]=ode45(@apolloeq,[0,20],x0); toc
   length(t), plot(y(:,1),y(:,3))
elapsed_time =
     0.3300
```

```
ans =
           689
```

得出的轨迹如图 3-9(a) 所示。



(a) 默认控制参数下的仿真结果 (错误结果)　　(b) 改变精度设置后的结果

图 3-9 不同精度要求下绘制的 Apollo 轨迹图

其实，这样直接得出的 Apollo 轨道是不正确的，因为这时 ode45() 函数选择的默认精度控制 RelTol 设置不恰当，可以减小该值，直至减小到 0.000001，则可以使用下面的语句进行仿真研究

```
>> options=odeset; options.RelTol=1e-6;
   tic, [t1,y1]=ode45(@apolloeq,[0,20],x0,options); toc
   length(t1), plot(y1(:,1),y1(:,3)),
elapsed_time =
    0.7010
ans =
          1873
```

将得出的轨迹如图 3-9(b) 所示。可见，在新的默认精度下结果是完全不同的。这时，再进一步减小精度控制误差限也不会有太大的改进了。所以在仿真结束后有时有必要减小精度误差限 RelTol，看看得出的结果是否还相同，依此判定这样选择的精度要求是否合适，否则对求解结果是否正确没有把握。

由 plot(t(1:end-1),diff(t)) 命令则可以绘制出计算步长的曲线，如图 3-10 所示。从得出的图形可以看出变步长算法的意义，即在需要小步长时取小步长，而变换缓慢时取大步长计算，这样就可以保证以高效率求解方程。

如果两个高阶微分方程都同时隐式地含有 $x^{(m)}$ 和 $y^{(n)}$ 项，则首先需要对之进行相应的处理，然后再用上述的方法进行最终变换。下面将通过一个例子加以说明。

【例 3.36】 假设系统模型以二元方程组形式给出

$$\begin{cases} \ddot{x}+2\dot{y}x=2\ddot{y} \\ \ddot{x}\dot{y}+3\dot{x}\ddot{y}+x\dot{y}-y=5 \end{cases}$$

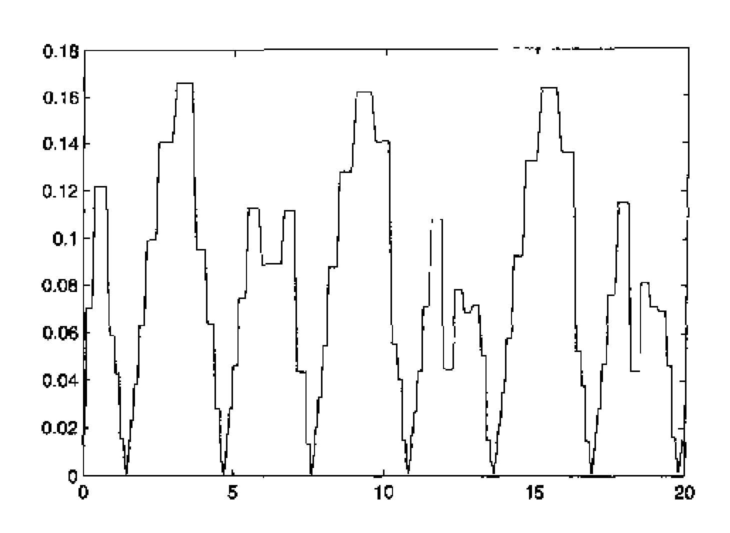


图 3-10 仿真过程中的计算步长

可见，这两个方程均同时含有 $\ddot{x}$ 和 $\ddot{y}$，所以，仍可以选择一组状态变量 $x_1 = x, x_2 = \dot{x}, x_3 = y, x_4 = \dot{y}$，我们的目的是先消去其中一个高阶导数，求解第一个式子，得出 $\ddot{y}$ 的解析表达式为

$$\ddot{y} = \dot{y}x + \frac{\ddot{x}}{2}$$

然后将它代入第二个式子中，则可以解出 $\ddot{x}$ 为

$$\ddot{x} = \frac{2y + 10 - 2x\dot{y} - 6x\dot{x}\dot{y}}{2\dot{y} + 3\dot{x}}$$

这样可以写出其状态方程表示为

$$\dot{x}_2 = \frac{2x_3 + 10 - 2x_1x_4 - 6x_1x_2x_4}{2x_4 + 3x_2}$$

将上面的结果再代入到 $\ddot{y}$ 的方程中，则

$$\dot{x}_4 = \frac{x_3 + 5 - x_1x_4 + 2x_1x_4^2}{2x_4 + 3x_2}$$

综上所述，可以列写出方程的一阶微分方程组表示为

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \dfrac{2x_3 + 10 - 2x_1x_4 - 6x_1x_2x_4}{2x_4 + 3x_2} \\ \dot{x}_3 = x_4 \\ \dot{x}_4 = \dfrac{x_3 + 5 - x_1x_4 + 2x_1x_4^2}{2x_4 + 3x_2} \end{cases}$$

事实上，这样的方程还是不太容易手工求解的，还可以依赖 MATLAB 的符号运算工具箱来求解该问题。为了方便求解起见，记 dx=$\ddot{x}$, dy=$\ddot{y}$，这样就能用下面的语句得出方程的解

```
>> syms x1 x2 x3 x4
   [dx,dy]=solve('dx+2*x4*x1=2*dy','dx*x4+3*x2*dy+x1*x4-x3=5','dx,dy');
   pretty(dx)
                    3 x4 x1 x2 + x4 x1 - x3 - 5
              -2 ---------------------------
                         2 x4 + 3 x2
>> pretty(dy)
                       2
                  2 x4  x1 - x4 x1 + x3 + 5
                  -------------------------
                         2 x4 + 3 x2
```

可见得出的结果与前面手工得出的完全一致。

对于更复杂的问题来说，手工变换的难度将很大，所以如果可能，可以采用计算机去求解有关方程，获得解析解。如果不能获得方程的解析解，也需要在描写一阶常微分方程组时列写出式子，得出问题的数值解。

3.4.6 微分代数方程的数值解法

在前面的介绍中，所介绍的常微分方程数值解法主要是针对能够转换成一阶常微分方程组的类型，假设其中的一些微分方程退化为代数方程，则用前面介绍的算法无法求解，必须借助微分代数方程的特殊解法。

所谓微分代数方程 (differential algebraic equation，简称 DAE)，是指在微分方程中，某些变量间满足某些代数方程的约束，所以这样的方程不能用前面介绍的常微分方程解法直接进行求解。假设微分方程的更一般形式可以写成

$$\boldsymbol{M}\dot{\boldsymbol{y}} = \boldsymbol{f}(t, \boldsymbol{y}) \tag{3.80}$$

描述 $\boldsymbol{f}(t, \boldsymbol{y})$ 的方法和普通常微分方程完全一致，而对真正的微分代数方程来说，$\boldsymbol{M}$ 矩阵为奇异矩阵，在微分代数方程求解程序中应该由求解选项中的 Mass 来表示该矩阵，考虑了这些因素则可以立即求解方程的解了。遗憾的是，在 MATLAB 提供的微分代数方程求解函数中，只能考虑 $\boldsymbol{M}$ 为常数矩阵的情形，而类似于例 3.31 中的隐式微分方程则不能直接求解。

〖例 3.37〗 考虑下面给出的微分代数方程

$$\begin{cases} \dot{x}_1 = -0.2x_1 + x_2x_3 + 0.3x_1x_2 \\ \dot{x}_2 = 2x_1x_2 - 5x_2x_3 - 2x_2^2 \\ x_1 + x_2 + x_3 - 1 = 0 \end{cases}$$

并已知初始条件为 $x_1(0) = 0.8, x_2(0) = x_3(0) = 0.1$。可以看出，最后的一个方程为代数方程，可

以视之为三个状态变量间的约束关系。用矩阵的形式可以表示该微分代数方程

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 0\end{bmatrix}\begin{bmatrix}\dot{x}_1\\\dot{x}_2\\\dot{x}_3\end{bmatrix}=\begin{bmatrix}-0.2x_1+x_2x_3+0.3x_1x_2\\2x_1x_2-5x_2x_3-2x_2^2\\x_1+x_2+x_3-1\end{bmatrix}$$

这样就可以写出相应的 MATLAB 函数

```
function dx=c3eqdae(t,x)
dx=[-0.2*x(1)+x(2)*x(3)+0.3*x(1)*x(2);
    2*x(1)*x(2)-5*x(2)*x(3)-2*x(2)*x(2);
    x(1)+x(2)+x(3)-1];
```

可以将 $\boldsymbol{M}$ 矩阵输入给 MATLAB 工作空间，并在命令窗口中给出如下命令[①]

```
>> M=[1,0,0; 0,1,0; 0,0,0]; options=odeset; options.Mass=M;
   x0=[0.8; 0.1; 0.1]; [t,x]=ode15s(@c3eqdae,[0,20],x0,options);
   plot(t,x)
```

由上面的语句可以得出此微分代数方程的解，如图 3-11 所示。

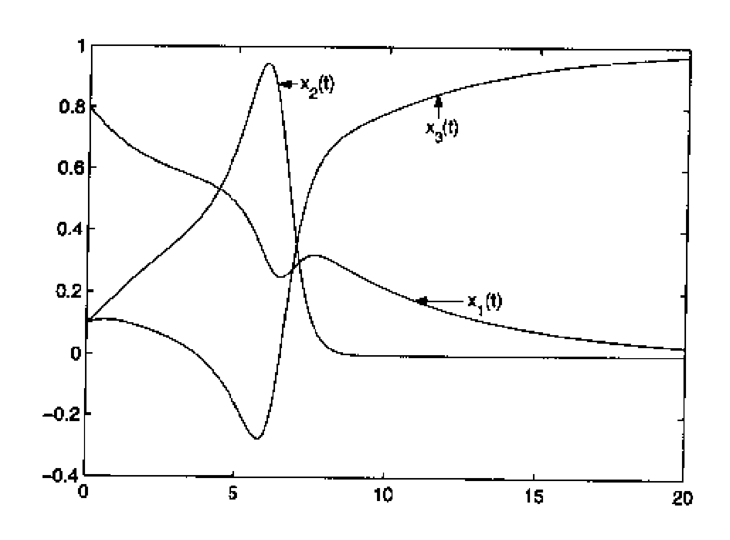


图 3-11　微分代数方程的数值解

事实上，一般的微分代数方程可以转换成常微分方程求解，例如在本例中，可以从约束式子中求出 $x_3(t)=1-x_1(t)-x_2(t)$，将其代入其他两个微分方程式子，则有

$$\begin{cases}\dot{x}_1=-0.2x_1+x_2[1-x_1(t)-x_2(t)]+0.3x_1x_2\\\dot{x}_2=2x_1x_2-5x_2[1-x_1(t)-x_2(t)]-2x_2^2\end{cases}$$

根据该方程可以写出

```
function dx=c3eqdae1(t,x)
dx=[-0.2*x(1)+x(2)*(1-x(1)-x(2))+0.3*x(1)*x(2);
    2*x(1)*x(2)-5*x(2)*(1-x(1)-x(2))-2*x(2)*x(2)];
```

[①] 此问题如果采用 ode45() 函数则将得出错误结果。

则用下面的命令就可以求解上述微分方程，从而最终得出原微分代数方程的解，所得出的解与前面的直接解法得出的完全一致。

```
>> x0=[0.8; 0.1]; [t1,x1]=ode45(@c3eqdae1,[0,20],x0);
   plot(t1,x1,t1,1-sum(x1'))
```

注意，这里如果使用 ode45() 函数也不会出现求解错误。

3.4.7 二阶微分方程边值问题的数值解法

二阶微分方程的边值问题的数学描述为

$$y''(x)=F(x,y,y') \tag{3.81}$$

假设想在区间 $[a,b]$ 上研究该方程的解，且已知在这两个边界点上满足

$$\alpha_a y(a)+\beta_a y'(a)=\gamma_a,\quad \alpha_b y(b)+\beta_b y'(b)=\gamma_b \tag{3.82}$$

上面的方程称为边界条件。

显然，利用前面的初值问题算法是不能直接使用的，因为并不能直接获得在初始时刻的各个变量的值。下面将讨论各种边值问题的数值解法。

3.4.7.1 线性方程边值问题的打靶算法

首先考虑一种简单的情况，即线性微分方程的边值问题求解的数值算法，考虑下面给出的微分方程

$$y''(x)+p(x)y'(x)+q(x)y(x)=f(x) \tag{3.83}$$

其中 $p(x)$, $q(x)$ 和 $f(x)$ 均为给定函数，而式 (3.82) 可以进一步简化成

$$y(a)=\gamma_a,\ y(b)=\gamma_b \tag{3.84}$$

下面将介绍的算法又称为打靶算法 (shooting method)，其基本想法是找出能够满足式 (3.82) 边值的相应初值 $y(0)$ 和 $y'(0)$，然后在利用前面介绍的初值算法去求解这一初值问题。

打靶算法可以由下面步骤完成：

(1) 求出下面方程初值问题的数值解 $y_1(b)$

$$y_1''(x)+p(x)y_1'(x)+q(x)y_1(x)=0,\ y_1(a)=1,\ y_1'(a)=0 \tag{3.85}$$

(2) 求出下面方程初值问题的数值解 $y_2(b)$

$$y_2''(x)+p(x)y_2'(x)+q(x)y_2(x)=0,\ y_2(a)=0,\ y_2'(a)=1 \tag{3.86}$$

(3) 求出下面方程初值问题的数值解 $y_p(b)$

$$y_p''(x)+p(x)y_p'(x)+q(x)y_p(x)=f(x),\ y_p(a)=0,\ y_p'(a)=1 \tag{3.87}$$

(4) 若 $y_2(b) \neq 0$，则计算

$$m = \frac{\gamma_b - \gamma_a y_1(b) - y_p(b)}{y_2(b)} \tag{3.88}$$

(5) 计算下面初值问题的数值解，则 $y(x)$ 即为原边值问题的数值解

$$y''(x) + p(x)y'(x) + q(x)y(x) = f(x),\ y(a) = \gamma_a,\ y'(a) = m \tag{3.89}$$

(6) 求解前面问题的数值解则应该首先得出一阶微分方程组模型，即取变量 $x_1 = y$, $x_2 = y'$，则可以得出

$$\frac{\mathrm{d}x_1}{\mathrm{d}x} = x_2,\ \ \frac{\mathrm{d}x_2}{\mathrm{d}x} = -q(x)x_1 - p(x)x_2 + f(x) \tag{3.90}$$

上面算法的 MATLAB 实现为

```
function [t,y]=shooting(f1,f2,tspan,x0f,varargin)
t0=tspan(1); tfinal=tspan(2);
ga=x0f(1); gb=x0f(2);
[t,y1]=ode45(f1,tspan,[1;0],varargin);
[t,y2]=ode45(f1,tspan,[0;1],varargin);
[t,yp]=ode45(f2,tspan,[0;0],varargin);
m=(gb-ga*y1(end,1)-yp(end,1))/y2(end,1);
[t,y]=ode45(f2,tspan,[ga;m],varargin);
```

在该函数中，`tspan` 为初始和终止仿真时间构成的向量，`x0f`=$[\gamma_a, \gamma_b]$ 为边界值。除此之外，用户需要定义两个辅助函数 `f1()`, `f2()` 来描述原模型，其中 `f1()` 描述

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -q(x_1)x_1 - p(x_1)x_2 + f(x_1) \end{cases} \tag{3.91}$$

而 `f2()` 描述原来方程的齐次部分

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -q(x_1)x_1 - p(x_1)x_2 \end{cases} \tag{3.92}$$

可以看出，在求解其中的初值问题时直接采用了 `ode45()` 函数，并允许使用该函数的任何附加选项，这些选项可以由 `varargin` 函数传给 `ode45()` 函数。如果您觉得对某些特定的问题 `ode45()` 函数不适合，则可以用其他函数名取代之。下面将通过简单例子介绍该函数的使用。

〖例 3.38〗 给定线性微分方程

$$y'' - 3y' + 2y = x,\ y(0) = 1,\ y(1) = 2$$

求其在 $[0, 1]$ 段方程的数值解。

由给出的方程，则可以立即写出下面两个 M 函数，其中 c3fun1.m 为

```
function xdot=c3fun1(t,x)
xdot=[x(2);
      -2*x(1)+3*x(2)];
```

而 c3fun2.m 为

```
function xdot=c3fun2(t,x)
xdot=[x(2);
      t-2*x(1)+3*x(2)];
```

采用下面的语句就可以求出原方程的解，并将其绘制出来，如图 3-12 所示。可见得出的 $x_1(t)$ 函数满足给定的边界条件。

```
>> [t,y]=shooting(@c3fun1,@c3fun2,[0,1],[1,2]); plot(t,y)
```

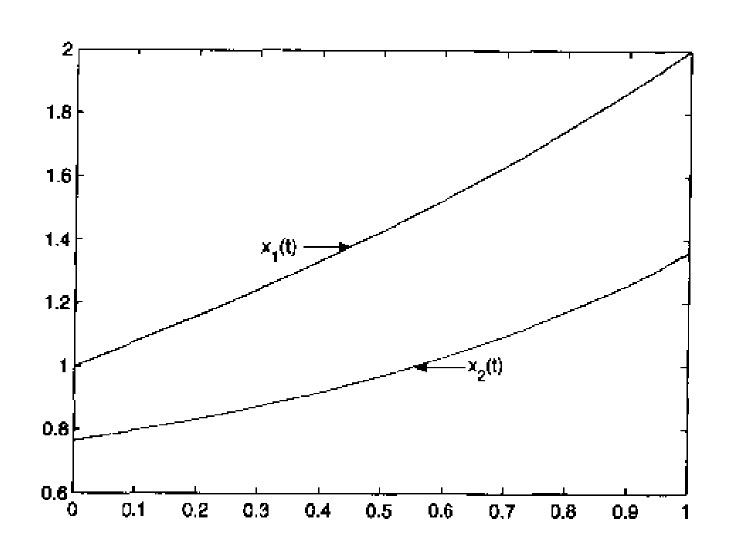


图 3-12 线性两点边值问题的解

其实，对这样简单的问题还可以由微分方程理论得出原方程的解析解为

$$y(x)=\frac{(\mathrm{e}^2-3)\mathrm{e}^x+(3-\mathrm{e})\mathrm{e}^{2x}}{4\mathrm{e}(\mathrm{e}-1)}+\frac{3}{4}+\frac{1}{2}x$$

可以由下面的语句求出上面得出结果的精度

```
>> y0=((exp(2)-3)*exp(t)+(3-exp(1))*exp(2*t))/(4*exp(1)*(exp(1)-1))+3/4+t/2;
   norm(y(:,1)-y0)  % 整个解函数检验
ans =
    4.4790e-008
>> norm(y(end,1)-2) % 终点条件检验
ans =
    2.2620e-008
```

由检验结果可见，这样求出的解精度还是较高的。

3.4.7.2 非线性方程边值问题的打靶算法

前面针对一类特殊的微分方程 (即线性微分方程) 边值问题进行了探讨，这种方法并不能求解式 (3.81) 和 (3.82) 所描述的一般问题，这里给出对此一般问题的数值解法。

假定原始问题可以转换成下面的初值问题

$$y'' = F(x, y, y'),\ y(a) = \gamma_a,\ y'(a) = m \tag{3.93}$$

则问题转换成求解 $y(m; b) = \gamma_b$，可以采用下面的 Newton 迭代法来求取 m

$$m_{i+1} = m_i - \frac{y(m_i; b) - \gamma_b}{(\partial y/\partial m)(m_i; b)} = m_i - \frac{v_1(b) - \gamma_b}{v_3(b)} \tag{3.94}$$

式中 $v_1 = y(m_i; x), v_2 = y'(m_i; x), v_3 = (\partial y/\partial m)(m_i; x), v_4 = (\partial y/\partial m)'(m_i; x)$，且可以由下面的微分方程初值问题来求解

$$\begin{cases} v_1' = v_2,\ \ v_1(a) = \gamma_a \\ v_2' = F(x, v_1, v_2),\ \ v_2(a) = m \\ v_3' = v_4,\ \ v_3(a) = 0 \\ v_4' = \dfrac{\partial F}{\partial y}(x, v_1, v_2)v_3 + \dfrac{\partial F}{\partial y'}(x, v_1, v_2)v_4,\ \ v_4(a) = 1 \end{cases} \tag{3.95}$$

其中要求能显式地求出 $\partial F/\partial y$, $\partial F/\partial y'$。在具体计算中可以指定一个 m 值，然后求解式 (3.95) 中的初值问题，将结果代入式 (3.94) 迭代一步，并将结果代入式 (3.95) 重新计算，直至两次计算出来的 m 值的误差在允许的范围内，则可以采用此 m 值，最后代入式 (3.93) 来求解原始问题。上面算法的 MATLAB 实现为

```
function [t,y]=nlbound(funcn,funcv,tspan,x0f,tol,varargin)
t0=tspan(1);tfinal=tspan(2);
ga=x0f(1); gb=x0f(2);
m=1; m0=0;
while (norm(m-m0)>tol), m0=m;
   [t,v]=ode45(funcv,tspan,[ga;m;0;1],varargin);
   m=m0-(v(end,1)-gb)/(v(end,3));
end
[t,y]=ode45(funcn,tspan,[ga;m],varargin);
```

其中用户必须自己编写一个 funcv() 函数来描述式 (3.95) 这的初值问题，下面将通过例子来演示此算法。

【例 3.39】 考虑下面的非线性微分方程边值问题

$$y'' = F(x, y, y') = 2yy',\ y(0) = -1,\ y(\pi/2) = 1$$

可以容易地求出下面偏导数 $\partial F/\partial y = 2y'$, $\partial F/\partial y' = 2y$，代入式 (3.95) 中的第 4 个式子可以立即得出 $\dot{v}_4 = 2x_2x_3 + 2x_1x_4$，故可以写出下面 MATLAB 函数

```
function xdot=c3fun3(t,x)
xdot=[x(2);
      2*x(1)*x(2);
      x(4);
      2*x(2)*x(3)+2*x(1)*x(4)];
```

而原微分方程模型的 MATLAB 表示可以写成

```
function xdot=c3fun4(t,x)
xdot=[x(2);
      2*x(1)*x(2)];
```

这时可以通过下面 MATLAB 语句来求解原始问题，得出的结果如图 3-13 所示，可见解出的 $x_1(t)$ 函数满足给定的边界要求。

```
>> [t,y]=nlbound(@c3fun4,@c3fun3,[0,pi/2],[-1,1],1e-8);
   plot(t,y); set(gca,'xlim',[0,pi/2]);
```

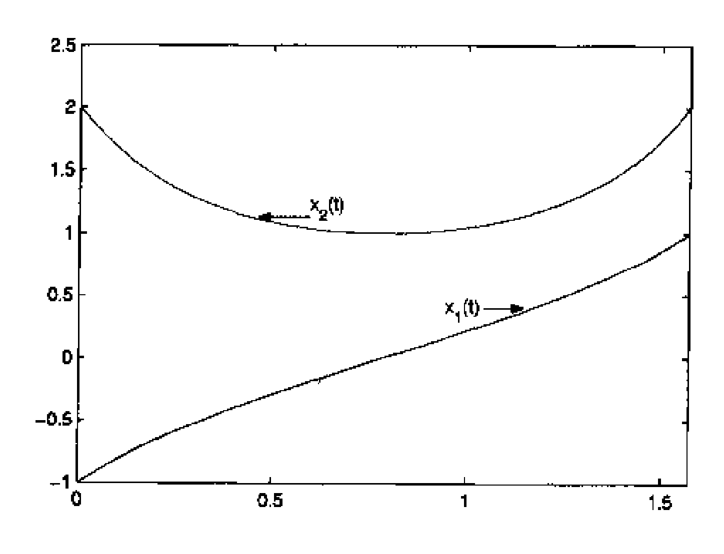


图 3-13 线性两点边值问题的解

该问题的解析解为 $y(x)=\tan(x-\pi/4)$，这样就可以由下面的语句比较得出的解是否精确了：

```
>> y0=tan(t-pi/4); norm(y(:,1)-y0)
ans = 2.6365e-007
>> norm(y(end,1)-1)
ans = 4.3509e-008
```

可见这样得出的数值解和解析解极其接近，且满足原始边值条件。

3.4.7.3 线性微分方程的有限差分算法

考虑式 (3.83) 中给出的线性微分方程，如果用前面介绍的中心差分数值微分公式取

代其中的微分项，则可以推导出下面的线性代数方程组

$$\begin{bmatrix} t_1 & v_1 & & & \\ w_2 & t_2 & v_2 & & \\ & \ddots & \ddots & \ddots & \\ & & w_{n-2} & t_{n-2} & v_{n-2} \\ & & & w_{n-1} & t_{n-1} \end{bmatrix} \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_{n-2} \\ \eta_{n-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-2} \\ b_{n-1} \end{bmatrix} \tag{3.96}$$

其中 n 为中间计算点的个数，$\eta_i, i=1,2,\cdots,n-1$ 为微分方程的数值解，即 $y_{i+1}=\eta_i$，而 $y_1=\gamma_a$ 和 $y_n=\gamma_b$，其他参数可以由下式计算出来

$$\begin{aligned} & t_i=-2+h^2q(x_i),\ v_i=1+\frac{h}{2}p(x_i) \\ & w_i=1-\frac{h}{2}p(x_i),\ b_i=h^2f(x_i) \end{aligned} \qquad i=1,\cdots,n-1 \tag{3.97}$$

且

$$b_1=b_1-w_1\gamma_a,\quad b_{n-1}=b_{n-1}-v_{n-1}\gamma_b \tag{3.98}$$

式中 h 为计算步长 $h=(b-a)/(n-1)$, $x_i=a+(i-1)h$。上面算法的 MATLAB 实现为

```
function [x,y]=fdiff(funcs,tspan,x0f,n)
t0=tspan(1);tfinal=tspan(2);
ga=x0f(1); gb=x0f(2);
h=(tfinal-t0)/n;
for i=1:n, x(i)=t0+h*(i-1); end, x0=x(1:n-1);
t=-2+h^2*feval(funcs,x0,2); tmp=feval(funcs,x0,1);
v=1+h*tmp/2; w=1-h*tmp/2; b=h^2*feval(funcs,x0,3);
b(1)=b(1)-w(1)*ga; b(n-1)=b(n-1)-v(n-1)*gb;
b=b'; A=diag(t);
for i=1:n-2, A(i,i+1)=v(i); A(i+1,i)=w(i+1); end
y=inv(A)*b;  x=[x tfinal]; y=[ga; y; gb]';
```

其中要求给定的函数为 funcs(x,key)，当 key=1,2,3 时分别表示 $p(x),q(x),f(x)$ 函数；另外，n 为要计算的点数，从算法来看，要求等间距的计算点分布，所以在解决一些复杂问题时，该算法不一定实用。这里将通过例子来演示该函数的使用。

〖例 3.40〗 考虑下面的线性微分方程边值问题

$$y''+(1+x)y'+(1-x)y=1+x^2,\ y(0)=1,\ y(1)=4$$

针对该微分方程可以编写下面的 MATLAB 函数

```
function y=c3fun5(x,key)
switch key
case 1, y=1+x;
case 2, y=1-x;
```

```
default, y=1+x.^2;
end
```

这时要计算出各个计算点处的 y 值，则可以由下面的方式调用 fdiff() 函数，绘制出 $y(t)$ 的曲线，如图 3-14 所示。

```
>> tic, [t,y]=fdiff(@c3fun5,[0,1],[1,4],50); toc
```

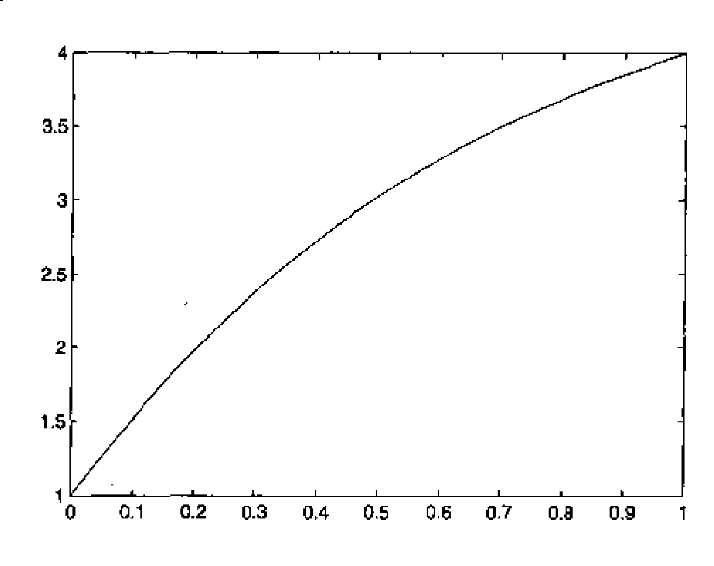


图 3-14 线性两点边值问题的解

当然，利用式 (3.96) 方程的三对角特性，可以采用 LU 分解来容易地求解，但这样会使得 MATLAB 编程变得更烦琐。从前面给出的算法看，只有第 2 种算法不要求原始函数为线性的，但该算法需要求出 $F(\cdot)$ 函数的偏导数，所以一般在线性方程求解时采用第 1 种方法，其他情况下可以采用第 2 种方法。

3.4.8 常微分方程的解析求解方法

在 MATLAB 的符号运算工具箱下，微分方程求解可以由 dsolve() 函数来完成，这里将通过下面的例子来演示微分方程解析解问题。

【例 3.41】 考虑下面的线性常微分方程

$$y^{(4)} - 4y^{(3)} + 8\ddot{y} - 8\dot{y} + 4y = 2$$

在符号运算工具箱中，用 'D4y' 字符串表示 y 变量的四阶导数，所以可以用下面的语句定义并求解此微分方程。

```
>> syms y; X=dsolve('D4y-4*D3y+8*D2y-8*Dy+4*y=2')
X =
   1/2+C1*exp(t)*cos(t)+C2*exp(t)*sin(t)+C3*exp(t)*sin(t)*t+
   C4*exp(t)*cos(t)*t
```

可以看出，只用了一个语句就求出了该微分方程的解析解。符号运算工具箱甚至能获得更高阶线性微分方程的“解析解”。

```
>> syms y; X1=dsolve('D6y + 7*D5y -3*D4y-4*D3y+8*D2y-8*Dy+4*y=2')
X1 =
   1/2+C1*exp(-7.3120156719066018548102886212053*t)+
   C2*exp(-1.4026118203972921605160158286315*t)+
   C3*exp((.15497080091830571520852952083837-
         .79814966819369424655113333983561*i)*t)+
   C4*exp((.15497080091830571520852952083837+
         .79814966819369424655113333983561*i)*t)+
   C5*exp((.70234294523364129245462270408004-
         .31097463157306344053848271228903*i)*t)+
   C6*exp((.70234294523364129245462270408004+
         .31097463157306344053848271228903*i)*t)
```

事实上，这仍不是解析解，而是更精确的数值解与解析解的混合体。尽管这不是数学意义上的解析解，但对一般的应用是足够了。

再考虑例 3.30 中给出的二阶非线性方程 —— Van der Pol 方程

$$\ddot{y}+\mu(y^2-1)\dot{y}+y=0$$

可以给出如下的命令

```
>> syms mu y; dsolve('D2y+mu*(y^2-1)*Dy+y=0')
Warning: Compact, analytic solution could not be found.
      It is recommended that you apply PRETTY to the output.
      Try mhelp dsolve, mhelp RootOf, mhelp DESol, or mhelp allvalues
      for more information.
> In C:\MATLAB6p1\toolbox\symbolic\dsolve.m at line 299
ans =
   '&where'(a,[{b(a)*diff(b(a),a)+mu*(a^2-1)*b(a)+a = 0}, {a = y(t),
    b(a) = diff(y(t),t)}, {y(t) = a, t = Int(1/b(a),a)+C1}])
```

由结果可知，对这样的非线性微分方程来说，符号运算工具箱是无能为力的了。所以对这类问题只能采用数值解的方法去研究。

〖例 3.42〗 简单方程的初值、边值问题解析解也能用符号运算工具箱直接求出，例 3.38 的问题重新在下面给出

$$y''-3y'+2y=x,\ \ y(0)=1,\ y(1)=2$$

则可以用下面的语句求出原始问题的解析解

```
>> f=dsolve('D2y-3*Dy+2*y=t','y(0)=1','y(1)=2');
   pretty(simple(f))
                          (exp(2) - 3) exp(t)         (-3 + exp(1)) exp(2 t)
 3/4 + 1/2 t - 1/4 ------------------- + 1/4 ----------------------
                           exp(1) - exp(2)               exp(1) - exp(2)
```

3.5 非线性方程与最优化问题求解

多元非线性方程组的求解和最优化问题往往需要调用最优化工具箱来解决。随着 MATLAB 6.1 版本的推出，最优化工具箱也升级成了 2.1.1 版。本书将以该版本为主介绍解方程和最优化问题。

最优化工具箱提供了 20 多个选项，用户可以输入 optimset 命令将它们显示出来。下面仅列出一些常用的选项：

- **Display 选项** 该选项决定函数调用时中间结果的显示方式，其中 'off' 为不显示，此选项值是默认的；'iter' 表示每步都显示；'final' 只显示最终结果。
- **GradConstr 选项** 表示是否存在约束函数的梯度，取值为 'off' 和 'on'。
- **GradObj 选项** 表示是否已知目标函数梯度，其中 'off' 为未知，而 'on' 为已知。
- **LargeScale 选项** 表示是否用大规模问题算法，取值为 'on' 或 'off'。在求解中小型问题时通常将该选项设置为 'off'。
- **LineSearchType 选项** 表示线性搜索类型，默认的 'quadcubic' 为四次法，而 'cubicpoly' 为三次多项式法。
- **MaxIter 选项** 最大允许迭代次数，默认为 400 次；选择空矩阵则表示取默认值。
- **TolFun 选项** 目标函数误差容限，选择空矩阵则表示默认值 10^{-6}。
- **TolX 选项** 自变量误差容限，选择空矩阵则表示默认值 10^{-6}。

可以先用 OPT=optimset 命令来调入一组默认选项值，如果想改变其中某个参数，则可以调用 optimset() 函数完成，或更直观地，用结构体属性的方式设置新参数。例如，不求解大规模问题时最好用下面的语句关闭大规模问题解法选项：

```
>> OPT=optimset; OPT.LargeScale='off';
```

这样可以将 LargeScale 选项设为 'off'。

3.5.1 非线性方程求解

在 MATLAB 的最优化工具箱中提供了多元方程的求解函数 fsolve()，该函数的基本调用格式为：

```
[x,yfun,key,c]=fsolve(F,x0,OPT);
```

其中 F 为要求解问题的数学描述，它可以是一个 MATLAB 函数，也可以是一个函数句柄；x0 为自变量的起始搜索点；OPT 为最优化工具箱的选项设定；x 为返回的解；而 yfun 是原函数在 x 点处的值。函数名是由引号括起来的，该函数为非线性方程组的一般描述，而初值为求根过程的起始点。返回的 key 表示函数返回的条件，1 表示已经求解出方程的解，而 0 表示未搜索到方程的解。返回的 c 为解的附加信息，该变量为一个结构体变量，其 iterations 成员变量表示迭代的次数，而 funcCount 是目标函数的调用次数。

〖例 3.43〗 考虑下面的二元方程

$$\begin{cases} x^2+y^2-1=0 \\ 0.75x^3-y+0.9=0 \end{cases}$$

可以编写出一个描述此二元方程的函数 my2deq()，其内容为

```
function q = my2deq(p)
q=zeros(2,1);
q(1)=p(1)*p(1)+p(2)*p(2)-1;  q(2)=0.75*p(1)^3-p(2)+0.9;
```

这样就可以在给定的初值 $x_0=1, y_0=2$ 下调用 fsolve() 函数，直接求出方程的根

```
>> OPT=optimset; OPT.LargeScale='off';
   [x,Y,c,d] = fsolve(@my2deq,[1; 2],OPT),
x =
    0.3570
    0.9341
Y =
  1.0e-008 *
   -0.9431
    0.1248
c =
     1
d =
       iterations: 5
        funcCount: 28
         stepsize: 1.0028
     cgiterations: []
    firstorderopt: []
        algorithm: 'medium-scale: Gauss-Newton, line-search'
```

可以看出，在求解此二元方程时仅调用了方程函数 28 次就得出了方程的解，把解回代到原始方程中也得到了较高精度的结果 (1.3×10^{-10})。

同样，可以建立如下的 inline() 函数，由下面的语句也可以得出相同的结果。

```
>> f=inline('[p(1)*p(1)+p(2)*p(2)-1; 0.75*p(1)^3-p(2)+0.9]','p');
   [x,Y] = fsolve(f,[1; 2],OPT)
x =
    0.3570
    0.9341
Y =
  1.0e-008 *
   -0.9431
    0.1248
```

可见，引入了 inline() 函数，无需为要求解的每个数学问题都编写一个单独的 MATLAB 模型文件，这样使得问题的求解与文件管理变得更容易、方便。

若改变初始猜测值，令 x0=[-1;0]，则

```
>> [x,Y,c,d] = fsolve(f,[-1,0]',OPT); x, Y, kk=d.funcCount
x =
   -0.9817
    0.1904
Y =
  1.0e-011 *
   -0.5339
   -0.1228
kk =
    27
```

可见，初值改变之后，还能得出另外一组解，所以说初值的选择有时对整个问题的求解有很大的影响，在某些初值下甚至无法搜索到方程的解。

观察原始方程，可以看出，从第 2 个方程可以解出 y，代入第 1 个方程，则可以得出一个一元 6 次的代数方程，该方程应该有 6 个根，可以为实数的，也可以为复数的。而 MATLAB 最优化工具箱中的函数只能解决实数解问题，再配以符号运算工具箱则可能解出所有的解。

通常，和常微分方程求解一样，最优化工具箱中每个求解函数都允许在选项 OPT 后面附加若干个其他参数，而这些参数可以对等地在数学模型函数中定义，但顺序必须和求解函数中的完全一致。

另外，使用 MATLAB 的符号运算工具箱往往能解出更精确的代数方程的根。在 MATLAB 下可以给出如下的命令

```
>> syms x y;
   X=solve('x^2+y^2-1=0','75*x^3/100-y+9/10=0')
X =
    x: [6x1 sym]
    y: [6x1 sym]
```

显然，这样得出的方程阶次太高，不能获得解析解。然而，利用 MATLAB 的符号运算工具箱可以得出原始问题的高精度数值解为

```
>> x1=X.x
x1 =
  [                                            -.98170264842676789676449828873194]
  [-.55395176056834560077984413882735-.35471976465080793456863789934944*i]
  [-.55395176056834560077984413882735+.35471976465080793456863789934944*i]
  [                                             .35696997189122287798839037801365]
  [ .86631809883611811016789809418650-1.2153712664671427801318378544391*i]
  [ .86631809883611811016789809418650+1.2153712664671427801318378544391*i]
```

```
>> y1=X.y
y1 =
  [                                     .19042035099187730240977756415289]
  [ .92933830226674362852985276677202-.21143822185895923615623381762210*i]
  [ .92933830226674362852985276677202+.21143822185895923615623381762210*i]
  [                                     .93411585960628007548796029415446]
  [-1.4916064075658223174787216959259-.70588200721402267753918827138837*i]
  [-1.4916064075658223174787216959259+.70588200721402267753918827138837*i]
```

可以看出，除了前面得出的两组实数根外，还得出了另外 4 组复数根，这是用普通数值解法所得不出来的。下面验证一下这样得出的根是不是原方程的根。若取第 2 对根代入原方程中，则

```
>> x=x1(2); y=y1(2); [eval('x^2+y^2-1') eval('75*x^3/100-y+9/10')]
ans =
   0     0
```

然而，解析求解的方法并不是万能的，因为这里的例子最终可以转换为一元高次代数方程，所以能用它求解，但更一般的方程是不能解出的。

〖例 3.44〗 下面考虑一个二元非线性方程组

$$x^2\sin(y) = x+1,\ \ y = x\cos(0.1x^2+3x)+0.5$$

用符号运算工具箱的 solve() 函数和最优化工具箱的 fsolve() 函数一般只能求解出该方程的一个根：

```
>> X=solve('x^2*sin(x)=x+1','y=x*cos(0.1*x^2+3*x)+0.5');
   [X.x; X.y] % 显示 x,y 的结果
ans =
  [ -.65380805760604745254428894837243+.59926044432990173736831588635030*i]
  [ -.29577108370681844653922317683028-2.2837911181207345106979215645352*i]
```

事实上，若给出下面的 MATLAB 语句

```
>> ezplot('x^2*sin(y)-x-1'); hold on % 绘制第一条曲线并保持屏幕
   ezplot('-y+x*cos(0.1*x^2+3*x)+0.5')  % 绘制第二条曲线
```

则可以用图解法表示出该方程根的位置，如图 3-15 所示。其中用曲线编辑的方法将第二个方程的曲线转换成了虚线。可以看出，在当前区域内两条曲线共有 19 个交点，这些点均是原方程组的实根。用符号运算工具箱只能求出一个根，且为复数根，不在这些根内。如果扩大图示区域显然将得到更多的实根，同时还可能有大量复数根 (事实上应该有无穷多根)。所以在方程求根过程中不能过分依赖数值方法和解析方法，有时还应该适当考虑图解的方法。

3.5.2 无约束最优化问题求解

无约束最优化问题的一般描述为

$$\min_{\boldsymbol{x}} F(\boldsymbol{x}) \tag{3.99}$$

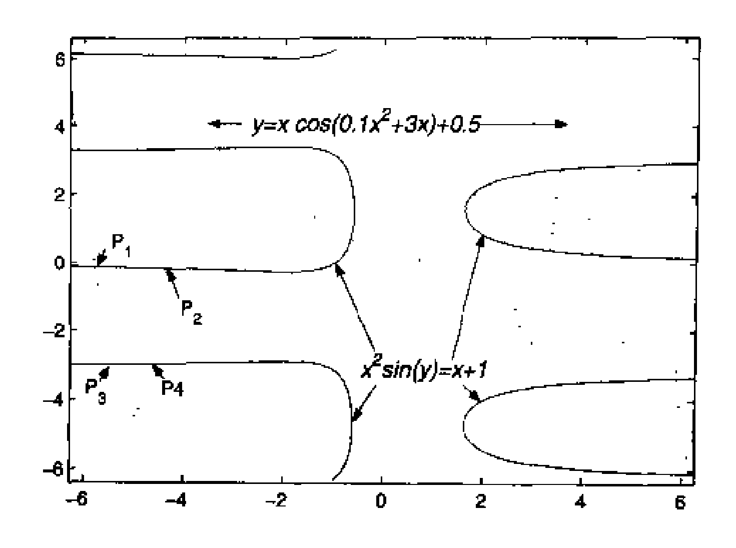


图 3-15 方程求根的图解法

其中 $\boldsymbol{x}=[x_1,x_2,\cdots,x_n]^T$，该数学表示的涵义亦即求取一组 $\boldsymbol{x}$ 向量，使得最优化目标函数 $F(\boldsymbol{x})$ 为最小，故这样的问题又称为最小化问题。其实，最小化是最优化问题的通用描述，它不失普遍性。如果要求解最大化问题，那么只需给目标函数 $F(\boldsymbol{x})$ 乘一个负号就能立即将原始问题转换成最小化问题。

最优化工具箱中提供了基于文献 [37] 中给出的单纯形算法求解无约束最优化的函数 fminsearch()，该函数的调用格式为：

```
[x,fopt,key,c]=fminsearch(F, x0, OPT)
```

其输入与返回参数的定义与 fsolve() 一致。我们将通过以下的例子来演示无约束最优化问题的数值解法。

【例 3.45】 考虑下面的无约束最优化问题

$$\min_{\boldsymbol{x}}(3x_1^2-2x_1x_2+x_2^2+4x_1+3x_2)$$

其目标函数的 MATLAB 表示为

```
function y=opt_fun0(x)
y=3*x(1)*x(1)-2*x(1)*x(2)+x(2)*x(2)+4*x(1)+3*x(2);
```

编写了这样的函数后，就可以由下面的命令求解最优化问题。

```
>> format long; [x,f_opt,c,d]=fminsearch(@opt_fun0,[-1.2, 1]);
   x, f_opt, kk=d.funcCount
Optimization terminated successfully:
x =
  -1.74997728682171  -3.25001058436666
f_opt =
  -8.37499999785950
kk =
```

```
    85
```

如果采用 inline() 函数形式，则可以由下面的命令得出同样的解。

```
>> f=inline('3*x(1)*x(1)-2*x(1)*x(2)+x(2)*x(2)+4*x(1)+3*x(2)','x');
   [x,f_opt]=fminsearch(f,[-1.2, 1])
Optimization terminated successfully:
x =
  -1.74997728682171  -3.25001058436666
f_opt =
  -8.37499999785950
```

事实上，用高等数学的知识不难求出本例子的解析解为 $(-7/4,-13/4)$，所以这里的算法求解精度不是很高。用下面的语句改变求解精度设置。

```
>> OPT=optimset; OPT.TolX=1e-10; OPT.LargeScale='off';
   [x,f_opt,c,d]=fminsearch(f,[-1.2, 1],OPT);
   x, f_opt, kk=d.funcCount
Optimization terminated successfully:
   the current x satisfies the termination criteria using OPTIONS.TolX
   of 1.000000e-010 and F(X) satisfies the convergence criteria using
   OPTIONS.TolFun of 1.000000e-004
x =
  -1.75000002204235  -3.25000002048471
f_opt =
  -8.37500000000000
kk =
   164
```

正如该函数指出的那样，当前的收敛条件采用了目标函数的 10^{-4} 约束。实验表明，再进一步减小误差容限也不能显著地改进解的精度。

3.5.3 线性规划问题

线性规划问题的数学描述为

$$\min_{\boldsymbol{x}\ \text{s.t.}\ \boldsymbol{A}\boldsymbol{x}\leq \boldsymbol{B}} \boldsymbol{f}^T\boldsymbol{x} \tag{3.100}$$

记号 s.t. 是英文 subject to 的缩写，表示满足后面的关系。约束条件还可以进一步细化为线性等式约束 $\boldsymbol{A}_{\rm eq}\boldsymbol{x}=\boldsymbol{B}_{\rm eq}$，线性不等式约束 $\boldsymbol{A}\boldsymbol{x}\leq\boldsymbol{B}$, $\boldsymbol{x}$ 变量的上界向量 $\boldsymbol{x}_M$ 和下界向量 $\boldsymbol{x}_m$，使得 $\boldsymbol{x}_m\leq\boldsymbol{x}\leq\boldsymbol{x}_M$。

对不等式约束来说，这里的标准型是 "$\leq$" 关系式，如果原问题中某个式子是 "$\geq$" 关系式，则在不等号两边同时乘以 -1 就可以转换成 "$\leq$" 关系式了。

在新版本的最优化工具箱中提供了求解线性规划问题的 linprog() 函数，该函数的

调用格式为：

```
[x,fopt,key,c]=linprog(f,A,B,Aeq,Beq,LB,UB,x0,OPT)
```

其中，f,A,B,Aeq,Beq,LB,UB 与前面约束与目标函数公式中的记号是完全一致的，x0 为初始搜索点。各个矩阵约束如果不存在，则应该用空矩阵来占位。OPT 为控制选项。最优化运算完成后，结果将在变量 x 中返回，最优化的目标函数将在 fopt 变量中返回。我们将通过下面的例子来演示线性规划的求解问题。

〖例 3.46〗 考虑下面的 4 元线性规划问题

$$\max_{\boldsymbol{x}\ \text{s.t.}\left\{\begin{array}{l}x_1+x_2+3x_3+x_4=6\\-2x_2+x_3+x_4\leq 3\\-x_2+6x_3-x_4\leq 4\\x_1,x_2,x_3,x_4\geq 0\end{array}\right.}\left[-x_1+2x_2-x_3+3x_4\right]$$

首先将之转换成最小化问题，将原目标函数乘以 -1，则目标函数将改写成 $x_1-2x_2+x_3-3x_4$。套用线性规划的格式则可以得出 $\boldsymbol{f}^T$ 向量为 $[1,-2,1,-3]$。

再分析约束条件，可见由最后一条可以写成 $0\leq x_i\leq\infty$。可以建立起等式约束条件的 $\boldsymbol{A}_{\rm eq}$ 和 $\boldsymbol{B}_{\rm eq}$ 矩阵为 $\boldsymbol{A}_{\rm eq}=[1,1,3,1]$，且 $\boldsymbol{B}_{\rm eq}=6$。另外，可以写出不等式约束为

$$\boldsymbol{A}=\begin{bmatrix}0&-2&1&1\\0&-1&6&-1\end{bmatrix},\ \boldsymbol{B}=\begin{bmatrix}3\\4\end{bmatrix}$$

再考虑 $\boldsymbol{x}$ 变量的上下限，可以输入如下的命令来求解此最优化问题。

```
>> f=[1,-2,1,-3]; Aeq=[1,1,3,1]; Beq=6;
   A=[0,-2,1,1; 0,-1,6,-1]; B=[3; 4]; LB=zeros(4,1); UB=[];
   [x,f_opt]=linprog(f,A,B,Aeq,Beq,LB,UB)
Optimization terminated successfully.
x =
    0.0000
    1.0000
    0.0000
    5.0000
f_opt =
  -17.0000
```

3.5.4 二次型规划问题

二次型规划的数学表示为

$$\min_{\boldsymbol{x}\ \text{s.t.}\ A\boldsymbol{x}\leq B}\left(\frac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x}+\boldsymbol{f}^T\boldsymbol{x}\right)\tag{3.101}$$

在新版本的最优化工具箱中提供了求解二次型规划问题的 quadprog() 函数，该函数的调用格式为：

```
[x,fopt,key,c]=quadprog(H,f,A,B,Aeq,Beq,LB,UB,x0,OPT)
```

【例 3.47】 考虑下面的四元二次型规划问题。

$$\min_{\boldsymbol{x}\ \text{s.t.}\begin{cases}x_1+x_2+x_3+x_4\leqslant 5\\3x_1+3x_2+2x_3+x_4\leqslant 10\\x_1,x_2,x_3,x_4\geqslant 0\end{cases}}\left[(x_1-1)^2+(x_2-2)^2+(x_3-3)^2+(x_4-4)^2\right]$$

首先应该将原始问题写成二次型规划的模式，并展开目标函数，

$$f(x)=x_1^2-2x_1+1+x_2^2-4x_2+4+x_3^2-6x_3+9+x_4^2-8x_4+16$$

因为目标函数中的常数对最优化结果没有影响，所以可以放心地略去。这样就可以将二次型规划标准型中的 $\boldsymbol{H}$ 矩阵和 $\boldsymbol{f}^T$ 向量写为

$$\boldsymbol{H}=\text{diag}([2,2,2,2]),\ \ \boldsymbol{f}^T=[-2,-4,-6,-8]$$

从而可以给出下列 MATLAB 命令来求解二次型最优化问题。

```
>> f=[-2,-4,-6,-8]; H=diag([2,2,2,2]);
   OPT=optimset; OPT.LargeScale='off'; % 不使用大规模问题求解
   A=[1,1,1,1; 3,3,2,1]; B=[5;10]; Aeq=[]; Beq=[]; LB=zeros(4,1);
   [x,f_opt]=quadprog(H,f,A,B,Aeq,Beq,LB,[],[],OPT)
Optimization terminated successfully.
x =
     0.0000
     0.6667
     1.6667
     2.6667
f_opt =
   -23.6667
```

套用二次型规划标准型时，一定要注意 $\boldsymbol{H}$ 矩阵的生成，因为在式 (3.101) 中有一个 1/2 项，所以在本例中，$\boldsymbol{H}$ 矩阵对角元素是 2，而不是 1。另外这里得出的目标函数实际上不是原始问题中的最优函数，因为人为地除去了常数项。将得出的结果再补上已经除去了的常数项，则可以求出原问题目标函数的值为 6.3333。

3.5.5 一般非线性规划问题求解

有约束非线性最优化问题的一般描述为

$$\min_{\boldsymbol{x}\ \text{s.t.}\ G(\boldsymbol{x})\leqslant 0} F(\boldsymbol{x}) \tag{3.102}$$

其中 $\boldsymbol{x}=[x_1,x_2,\cdots,x_n]^T$，该数学表示的涵义亦即求取一组 $\boldsymbol{x}$ 向量，使得函数 $F(\boldsymbol{x})$ 最小化，且满足约束条件 $G(\boldsymbol{x})\leq 0$。这里约束条件可以是很复杂的，它既可以是等式约束，也可以是不等式约束等。

约束条件还可以进一步细化为线性等式约束 $A_{\rm eq}\boldsymbol{x}=\boldsymbol{B}_{\rm eq}$，线性不等式约束 $A\boldsymbol{x}\leq\boldsymbol{B}$，$\boldsymbol{x}$ 变量的上界向量 $\boldsymbol{x}_M$ 和下界向量 $\boldsymbol{x}_m$，使得 $\boldsymbol{x}_m\leq\boldsymbol{x}\leq\boldsymbol{x}_M$，还允许一般非线性函数的等式和不等式约束。

新版本的 MATLAB 最优化工具箱中提供了一个 fmincon() 函数，专门用于求解各种约束下的最优化问题。该函数的调用格式为：

[x,fopt,key,c]=fmincon(F,x0,$\boldsymbol{A}$,$\boldsymbol{B}$,$A_{\rm eq}$,$\boldsymbol{B}_{\rm eq}$, $\boldsymbol{x}_m$,$\boldsymbol{x}_M$,CFun,OPT);

其中，F 为给目标函数写的M 函数，x0 为初始搜索点。各个矩阵约束如果不存在，则应该用空矩阵来占位。CFun 为给非线性约束函数写的 M 函数，OPT 为控制选项。最优化运算完成后，结果将在变量 x 中返回，最优化的目标函数将在 fopt 变量中返回。选项有时是很重要的。

【例 3.48】 考虑下面的有约束最优化问题

$$\min_{\boldsymbol{x}\ \text{s.t.}\begin{cases}x_1^2+x_2^2+x_3^2-25=0\\8x_1+14x_2+7x_3-56=0\\x_1,x_2,x_3\geq 0\end{cases}}\left[1000-x_1^2-2x_2^2-x_3^2-x_1x_2-x_1x_3\right]$$

可以写出目标函数为

```
function y=opt_fun1(x)
y=1000-x(1)*x(1)-2*x(2)*x(2)-x(3)*x(3)-x(1)*x(2)-x(1)*x(3);
```

还可以写出非线性约束函数为

```
function [c,ceq]=opt_con1(x)
ceq=[x(1)*x(1)+x(2)*x(2)+x(3)*x(3)-25;
     8*x(1)+14*x(2)+7*x(3)-56];
c = [];
```

非线性约束函数返回变量分为 c 和 ceq 两个量，其中前者为不等式约束的数学描述，后者为非线性等式约束，如果某个约束不存在，则应该将其值赋为空矩阵。这样的约束函数处理比早期版本的工具箱中处理更方便、规范。

可以调用 fmincon() 函数求解此约束最优化问题

```
>> OPT=optimset; OPT.LargeScale='off'; x0=[1;1;1];
   LB=[0;0;0]; UB=[Inf; Inf; Inf]; A=[]; B=[]; Ae=[]; Be=[];
   [x,f_opt,c,d]=fmincon('opt_fun1',x0,A,B,Ae,Be,LB,UB,'opt_con1',OPT);
   x,f_opt,kk=d.funcCount
Optimization terminated successfully:
Magnitude of directional derivative in search direction
less than 2*options.TolFun and maximum constraint violation
```

```
 is less than options.TolCon
    Active Constraints:
         1
         2
x =
     3.5121
     0.2170
     3.5522
f_opt =
     961.7152
kk =
      58
```

考虑到第 2 个约束条件实际上是线性等式约束，故可以将非线性约束函数进一步简化为

```
function [c,ceq]=opt_con2(x)
ceq=x(1)*x(1)+x(2)*x(2)+x(3)*x(3)-25;
c = [];
```

这时，可以用下面的命令求解原始的最优化问题

```
>> x0=[1;1;1]; Ae=[8,14,7]; Be=56;
   [x,f_opt,c,d]=fmincon('opt_fun1',x0,A,B,Ae,Be,LB,UB,'opt_con2',OPT);
   x, f_opt, kk=d.funcCount
x =
     3.5121
     0.2170
     3.5522
f_opt =
     961.7152
kk =
      58
```

如果已知目标函数的偏导数向量 (或梯度)，则可以更容易地求解最优化问题。例如可以写出目标函数对 3 个自变量的偏导数为

$$\frac{\partial F}{\partial x_1}=-2x_1-x_2-x_3,\ \frac{\partial F}{\partial x_2}=-4x_2-x_1,\ \frac{\partial F}{\partial x_3}=-2x_3-x_1$$

这时可以将目标函数改写成

```
function [y,Gy]=opt_fun2(x)
y=1000-x(1)*x(1)-2*x(2)*x(2)-x(3)*x(3)-x(1)*x(2)-x(1)*x(3);
Gy=-[2*x(1)+x(2)+x(3); 4*x(2)+x(1); 2*x(3)+x(1)];
```

其中，Gy 表示原问题的 Jacob 矩阵。再调用最优化求解函数将得出下面的结果

```
>> LB=[0;0;0]; UB=[Inf; Inf; Inf]; A=[]; B=[]; Ae=[]; Be=[];
```

```
   OPT=optimset; OPT.GradObj='on';
   [x,f_opt,c,d]=fmincon('opt_fun2',x0,A,B,Ae,Be,LB,UB,'opt_con1',OPT);
   x,f_opt,kk=d.funcCount
x =
    3.5121
    0.2170
    3.5522
f_opt =
   961.7152
kk =
    25
```

可见，若已知目标函数的偏导数，则仅需 25 步就求出原问题的解。注意，若已知梯度函数，则应该将 GradObj 选项设置成 'on'，否则不能识别该梯度。

3.6 数据插值与统计分析

3.6.1 一维数据的插值拟合

插值问题的提出：

假设 $f(x)$ 是一维给定函数，且在相异的一组 n 个自变量 $x_1, x_2, \cdots, x_n$ 点处的值为 $f_1, f_2, \cdots, f_n$，则由这些已知点的信息获得该函数在其他点上值的方法称为函数的插值。如果在这些给定点的范围内进行插值，又称为内插，否则称为外插。如果从时间的概念上理解这个问题，则对 x_n 以后点的插值又称为预报。

MATLAB 语言中提供了若干个插值函数，如一维插值函数 **interp1()**，多项式拟合函数 **polyfit()** 等，还有大量的解决多维插值问题的函数。

一维插值问题可以由 interp1() 函数解决，该函数的调用格式为：

```
y1=interp1(x,y,x1,方法)
```

其中，x 和 y 两个向量分别表示给定的一组自变量和函数值数据，x1 为一组新的插值点，而得出的 y1 是在这一组插值点处的插值结果。插值方法一般可以选 'linear' (线性的，此选项是默认的，它在两个点间简单地采用直线拟合，故效果并不光滑)，'cubic' (三次的) 和 'spline' (样条型) 等，一般建议使用后两种。

函数 **polyfit()** 可以对给定已知点数据进行多项式拟合，该函数的调用格式为：

```
p=polyfit(x,y,n)
```

其中，x 和 y 两个向量分别表示给定的一组自变量和函数值数据，n 为预期的多项式阶次，返回的 p 为插值多项式系数。

除了这些插值函数之外，MATLAB 语言还提供了更专用的样条插值工具箱 (Spline

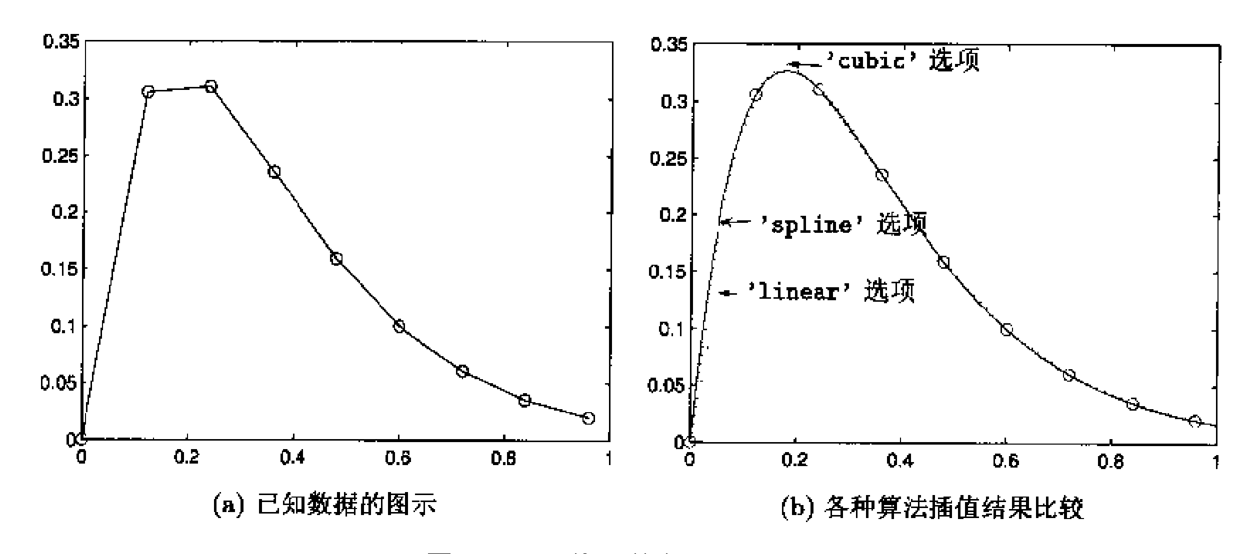
Toolbox) 可以直接应用。

〖例 3.49〗 假设已知的数据点来自下面的函数

$$f(x) = (x^2 - 3x + 5)\mathrm{e}^{-5x} \sin x$$

则可以由下面的语句生成数据，并绘制出数据的折线图，如图 3-16(a) 所示。

```
>> x=0:.12:1;
   y=(x.^2-3*x+5).*exp(-5*x).*sin(x); plot(x,y,x,y,'o')
```



(a) 已知数据的图示　　(b) 各种算法插值结果比较

图 3-16　一维函数各种插值结果

可以看出，由这样的数据直接连线绘制出来的曲线十分粗糙，可以再选择一组插值点，然后直接调用 interp1() 函数进行插值近似

```
>> x1=0:.02:1; y0=(x1.^2-3*x1+5).*exp(-5*x1).*sin(x1);
   y1=interp1(x,y,x1); y2=interp1(x,y,x1,'cubic');
   y3=interp1(x,y,x1,'spline');
   plot(x1,[y1',y2',y3'],':',x,y,'o',x1,y0)
   [max(abs(y0(1:49)-y2(1:49))),max(abs(y0-y3))]
ans =
     0.0244     0.0086
```

分别选择各种拟合选项，可以得出拟合结果与理论曲线，它们之间的比较如图 3-16(b) 所示。可以看出，默认的直线型拟合得到的曲线和图 3-16(a) 中的同样粗糙，因为该方法就是对各个点的直接连线。而用 'cubic' 和 'spline' 选项的拟合更接近于理论值。事实上，应用样条插值算法得出的插值十分逼近理论值，甚至用肉眼难以分辨。所以样条函数插值在一维数据插值拟合中还是很有效的。样条插值还可以通过 spline() 函数和样条插值工具箱求出。

还可以用以上给出的数据获得插值多项式

```
>> p1=polyfit(x,y,3),p2=polyfit(x,y,4),p3=polyfit(x,y,5)
p1 =
```

```
    3.2397   -5.3334    2.1408    0.0435
 p2 =
   -7.1411   16.9506  -13.5159    3.6779    0.0080
 p3 =
   10.7256  -32.8826   38.4877  -20.8059    4.5073    0.0009
```

再由上面得出的多项式求出对给定插值点的数据拟合

```
>> y1=polyval(p1,x1); y2=polyval(p2,x1); y3=polyval(p3,x1);
   x0=(x1.^2-3*x1+5).*exp(-5*x1).*sin(x1);
   plot(x1,y0,x1,y1,'--',x1,y2,':',x1,y3,'-.')
```

就可以得出的拟合曲线如图 3-17 所示。可见，三次多项式和四次多项式的拟合效果都不好，而 5 次多项式的拟合精度是很高的，可以满足要求。当 x 的值很大时，则这些多项式拟合的结果都不是很理想，所以如果不是特别要求拟合出数学模型，则没有必要采用多项式拟合，直接采用样条拟合就可以了。

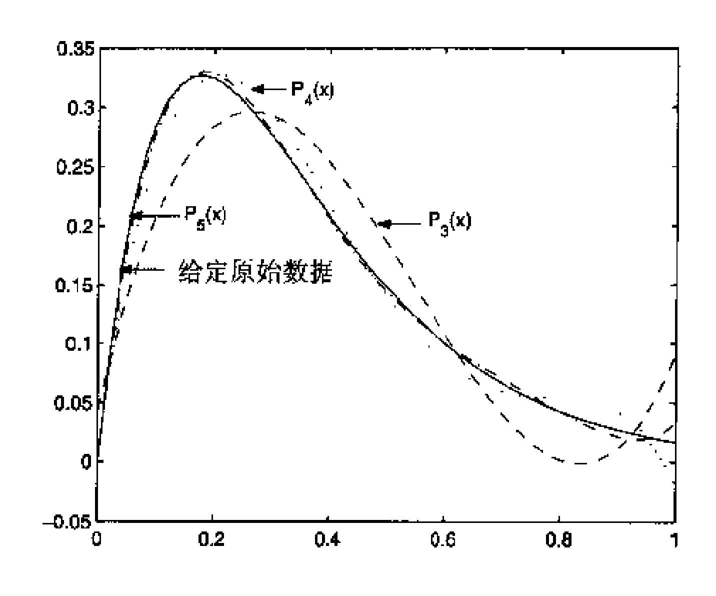


图 3-17 数据的多项式拟合结果

3.6.2 二维数据的插值拟合

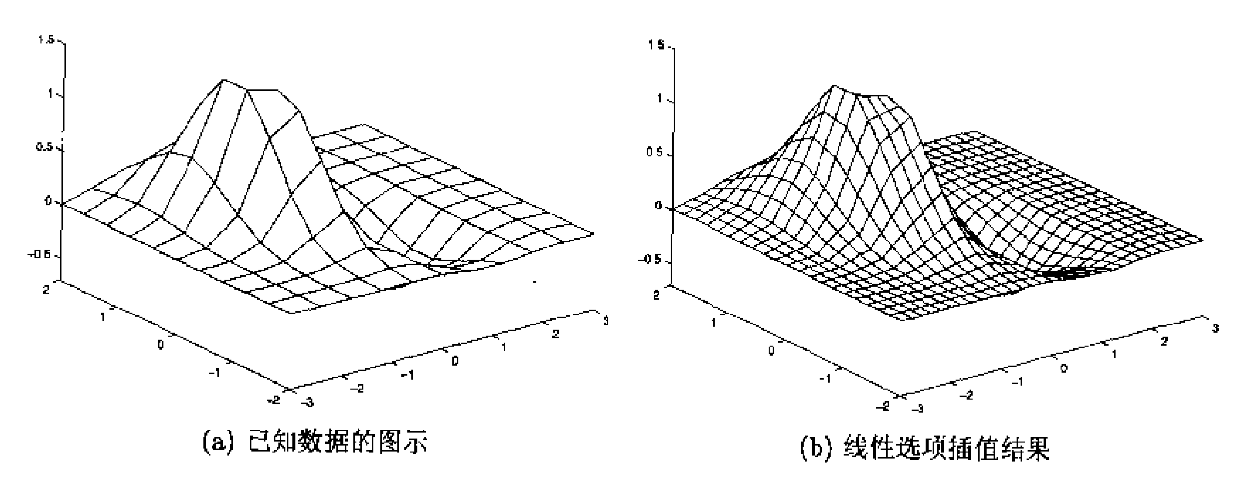
MATLAB 下提供了二维插值的函数，如 interp2()，该函数的调用格式为：

```
z1=interp2(x0,y0,z0,x1,y1,'方法')
```

其中 x0,y0,z0 为已知的数据，而 x1，y1 为插值点构成的新的网格参数，返回的 z1 矩阵为在插值网格点处的函数近似值。

〖例 3.50〗 回顾例 2.23 中的二元函数 $z = f(x,y) = (x^2 - 2x)\mathrm{e}^{-x^2-y^2-xy}$，假设仅知其中较少的数据，则可以由下面命令绘制出已知数据的网格图，如图 3-18(a) 所示。从图 3-18(a) 可以看出，由这些数据绘制的图形还是很粗糙的。

```
>> [x,y]=meshgrid(-3:.6:3, -2:.4:2);
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);
   mesh(x,y,z), axis([-3,3,-2,2,-0.7,1.5])
```



(a) 已知数据的图示　　(b) 线性选项插值结果

图 3-18 二维函数插值比较

选较密的插值点，则可以用下面的 MATLAB 语句采用默认的插值算法进行插值，得出的结果如图 3-18(b) 所示。

```
>> [x1,y1]=meshgrid(-3:.2:3, -2:.2:2);
   z1=interp2(x,y,z,x1,y1); mesh(x1,y1,z1), axis([-3,3,-2,2,-0.7,1.5])
```

可以看出，默认的线性插值方法还原后的三维表面图在很多地方还是太粗糙。可以用下面的命令分别由立方插值选项和样条插值选项来进行插值，得出的结果如图 3-19 所示。

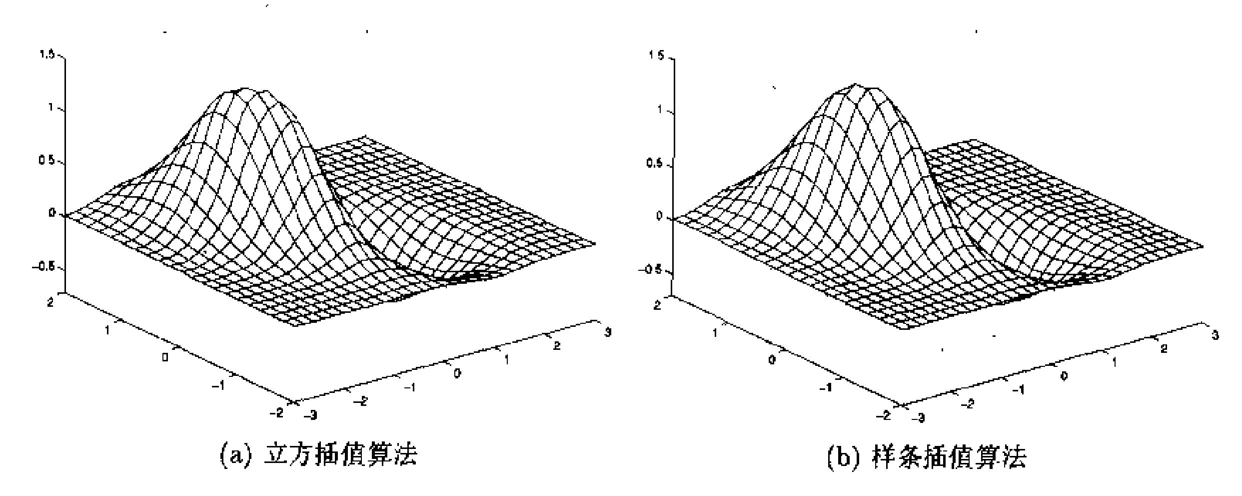
```
>> z1=interp2(x,y,z,x1,y1,'cubic'); z2=interp2(x,y,z,x1,y1,'spline');
   mesh(x1,y1,z1), axis([-3,3,-2,2,-0.7,1.5])
   figure; mesh(x1,y1,z2), axis([-3,3,-2,2,-0.7,1.5])
```

可以看出，这样的插值效果还是比较理想的。

通过上面的例子看，MATLAB 提供的二维插值函数还是能较好地进行二维插值运算的，但该函数有一个致命的问题，就是它只能处理以网格形式给出的数据，如果已知数据不是以网格形式给出的，则用该函数是无能为力的。在实际应用中大部分问题都是以实测的 (x_i, y_i, z_i) 点给出的，所以不能直接使用该函数进行二维插值。

3.6.3 最小二乘曲线拟合技术

假设有一组数据 $x_i, y_i, i = 1, 2, \cdots, N$，且已知这组数据满足某一函数原型 $\hat{y}(x) = f(\boldsymbol{a}, x)$，其中 $\boldsymbol{a}$ 待定系数向量，则最小二乘曲线拟合的目标就是求出这一组待定系数的



(a) 立方插值算法　　(b) 样条插值算法

图 3-19 二维函数其他插值结果比较

值，使得目标函数

$$J=\min_{\boldsymbol{a}}\sum_{i=1}^{N}[y_i-\hat{y}(x_i)]^2=\min_{\boldsymbol{a}}\sum_{i=1}^{N}[y_i-f(\boldsymbol{a},x_i)]^2 \tag{3.103}$$

为最小。在 MATLAB 的最优化工具箱中提供了 lsqcurvefit() 函数，可以解决最小二乘曲线拟合的问题，该函数的调用格式为：

```
[a,Jm]=lsqcurvefit(原型函数名, a0, x, y)
```

其中，a0 为最优化的初值，x，y 为原始输入输出数据向量，调用该函数则将返回待定系数向量 a，以及在此待定系数下的目标函数的值 Jm。

【例 3.51】 假设由下面的语句生成一组数据 x 和 y

```
>> x=0:.1:10;
   y=0.12*exp(-0.213*x)+0.54*exp(-0.17*x).*sin(1.23*x);
```

显然，可以知道该数据满足原型为

$$y(x)=a_1\mathrm{e}^{-a_2x}+a_3\mathrm{e}^{-a_4x}\sin(a_5x)$$

其中 a_i 为待定系数。采用最小二乘曲线拟合的目的就是获得这些待定系数，使得目标函数的值为最小。

根据已知的函数原型，可以编写出如下的 MATLAB 函数：

```
function y=c3xlsq(a,x)
y=a(1)*exp(-a(2)*x)+a(3)*exp(-a(4)*x).*sin(a(5)*x);
```

建立起函数的原型，则可以由下面的语句得出待定系数向量了。

```
>> [xx,res]=lsqcurvefit(@c3xlsq,[1,1,1,1,1],x,y)
Optimization terminated successfully:
```

```
 Relative function value changing by less than OPTIONS.TolFun
xx =
    0.1197    0.2125    0.5404    0.1702    1.2300
res =
    7.1637e-007
```

可以看出，这样得出的待定系数精度还算较高，接近与理论值 $a=[0.12,0.213,0.54,0.17,1.23]$。如果想进一步提高精度，则需要修改最优化的选项了，这时函数的调用格式也将发生变化：

```
>> f=optimset; f.TolFun=1e-10; % 修改精度限制
   [xx,res]=lsqcurvefit(@c3xlsq,[1,1,1,1,1],x,y,[],[],f)
Optimization terminated successfully:
 Relative function value changing by less than OPTIONS.TolFun
xx =
    0.1200    0.2130    0.5400    0.1700    1.2300
res =
    8.5516e-011
```

其中，两个空矩阵表示 a 向量的上下限，由于对这些参数的范围无限制，故采用了默认的表示形式。可以看出，修改误差限后，得出的拟合待定系数更加精确。

3.6.4 数据简单排序

假设给定了一个数据变量 X，那么就可以用 MATLAB 提供的各种函数对这样的数据进行处理了。例如若想求出这样一组数据的最大值和最小值，则可以分别采用下面的函数来求出：

```
[xM, i]=max(X); 或 [xm, i]=min(X);
```

其中返回的 xM 及 xm 分别为矩阵 X 各列的最大值或最小值所构成的向量，而 i 为各列最大值或最小值所在位置的行号构成的向量。这两个函数均可以只返回一个参数，而不返回 i。如果 X 是行向量或列向量，则得出的结果是该向量的最大值或最小值。

〖例 3.52〗 建立一个 10×10 的魔方矩阵，对之进行分析，最终可以得出如下的结果。

```
>> A=magic(10)
A =
    92    99     1     8    15    67    74    51    58    40
    98    80     7    14    16    73    55    57    64    41
     4    81    88    20    22    54    56    63    70    47
    85    87    19    21     3    60    62    69    71    28
    86    93    25     2     9    61    68    75    52    34
    17    24    76    83    90    42    49    26    33    65
    23     5    82    89    91    48    30    32    39    66
    79     6    13    95    97    29    31    38    45    72
```

```
    10    12    94    96    78    35    37    44    46    53
    11    18   100    77    84    36    43    50    27    59
>> [x_M,i]=max(A)
x_M =
    98    99   100    96    97    73    74    75    71    72
i =
     2     1    10     9     8     2     1     5     4     8
```

整个矩阵的最大值可以由 max(max(X)) 求出。

```
>> max(max(A))
ans =
   100
```

MATLAB 还提供了对给定向量的大小进行排序的函数 **sort()**，其调用格式和 **min()** 的几乎完全一致。调用了此函数之后，就可以将矩阵各列的值按照从小到大的顺序进行排列。辅以 MATLAB 编程的基本语句，就可以容易地得出从大到小的排序。

3.6.5 快速 Fourier 变换

离散数据 $x_i, i = 1, 2, \cdots, N$ 的 Fourier 变换是数字信号处理的基础，离散 Fourier 变换的数学表示为

$$X(k) = \sum_{i=1}^{N} x_i \mathrm{e}^{-2\pi \mathrm{j}(k-1)(i-1)/N}, \text{ 其中 } 1 \leq k \leq N \tag{3.104}$$

其逆变换定义为

$$x(k) = \frac{1}{N} \sum_{i=1}^{N} X(i) \mathrm{e}^{2\pi \mathrm{j}(k-1)(i-1)/N}, \text{ 其中 } 1 \leq k \leq N \tag{3.105}$$

快速 Fourier 变换 (FFT) 技术是求解离散数据 Fourier 变换的最实用的也是最通用的方法。MATLAB 中提供了内在函数 **fft()**，可以高效地求解 FFT 问题。该函数的另一个显著的特点是它可以对任意长度的向量进行变换，而不要求所变换的向量长度满足 2^n 约束，尽管满足这样长度的变换计算速度快些。

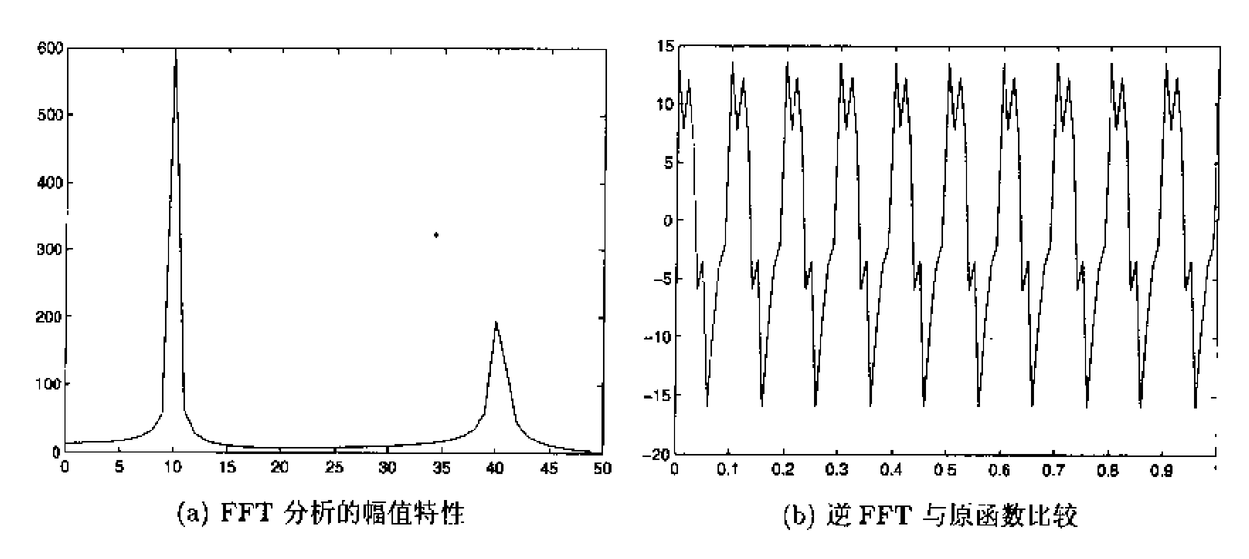
【例 3.53】 假设给定数学函数

$$x(t) = 12\sin(2\pi \times 10t + \pi/4) + 5\cos(2\pi \times 40t)$$

选择步长为 h，可以产生 L 个时间值 t_i，并求出这些点上的函数值为 x_i，其相应的频率点可以由 $f_0 = 1/h, 2f_0, 3f_0, \cdots$ 构成，然后可以由下面的语句

```
>> h=0.01; t=0:h:1;
   x=12*sin(2*pi*10*t+pi/4)+5*cos(2*pi*40*t);
   X=fft(x); f=t/h;
   plot(f(1:floor(length(f)/2)),abs(X(1:floor(length(f)/2))))
```

得出 FFT 幅值与频率的关系，如图 3-20 (a) 所示。这里仅取一半数据绘制图形的原因是为了避免众所周知的 FFT 分析的假频 (aliasing) 现象。从分析结果上可以看出，在幅值曲线上有两个峰值点，对应的频率值为 10Hz 和 40Hz，正是给定函数中的两个频率值。



(a) FFT 分析的幅值特性　　(b) 逆 FFT 与原函数比较

图 3-20　数据的 FFT 分析

快速 Fourier 逆变换可以由 ifft() 函数直接求解。

```
>> ix=real(ifft(X)); plot(t,x,t,ix,':')
   norm(x-ix)
ans =
   1.1635e-011
```

这样得出的逆 FFT 变换结果与原函数在图 3-20 (b) 中给出，可以看出二者完全一致。由于采用点较稀疏，故曲线看起来不是很光滑。

此外，MATLAB 还提供了二维或更高维的 FFT 与逆 FFT 函数，对二维问题可以调用 **fft2()** 和 **ifft2()** 函数，而高维问题可以使用 **fftn()** 和 **ifftn()** 函数。

3.6.6 数据分析与统计处理

- **伪随机数据生成与检验**

在系统仿真领域经常要用到随机数据，随机数的生成通常有两类方法，其一是依赖一些电子元件发出随机信号，这种方法又称为物理生成法；另一类是通过数学的算法，仿照随机数发生的规律计算出随机数，由于产生的随机数是由数学公式计算出来的，所以这类随机数又称为“伪随机数”。

伪随机数至少有下面两个优点：首先，这样随机数是可以重复的，这样就创造了重复实验的条件；另外，随机数满足的统计规律可以人为地选择，例如可以自由地选择均匀分布、正态分布、Poisson 分布等，来满足我们的需要。

均匀分布随机数：MATLAB 中提供了可靠的 [0,1] 区间上均匀分布随机数生成函数 rand()，该函数的调用格式为：

```
x=rand(n,m);
```

其中 n 和 m 为想生成随机数矩阵的行数和列数，如果只想生成一个均匀分布的随机数列向量，则用 x=rand(n,1) 即可。该函数还可以用于生成多维随机数组，例如使用命令 x=rand(n,m,k)。在算法上采用了较新的多种子的随机数发生算法，用 s=rand('state') 命令可以获得所使用的种子向量，如果想将随机数发生程序的种子设置为 s，则可以使用 rand('state',s)，这样就能保证重复得到所需的随机数。

假设用户得到了一组满足 [0,1] 区间上均匀分布的随机数 x_i，则若想获得在任意的 $[a,b]$ 区间上均匀分布的随机数，只需用 $y_i = a + (b-a)x_i$ 变换即可。

〖例 3.54〗 利用 MATLAB 提供的函数生成 30000 个均匀分布的随机数，然后检验其随机数的指标，例如均值、方差等，可以给出如下的一些命令

```
>> x=rand(30000,1); y=x(find(x>=0.5)); format long
   [mean(x), length(y)/length(x), min(x), max(x)]
ans =
   0.50263676044190   0.50446666666667   0.00001558352099   0.99998898980467
```

可以看出，利用此函数构成的随机数的均值接近 0.5，且大于 0.5 的随机数个数接近总数的 50%，生成的随机数最大值接近 1，最小值接近于 0，所以说这样生成的伪随机数还是较理想的。

正态分布随机数：满足标准正态分布的随机数 $N(0,1)$ 可以由 randn() 函数得出，其调用格式与 rand() 完全一致，但产生的是均值为 0，方差为 1 的正态分布随机数 $N(0,1)$。假设已经获得了标准正态分布随机数 x_i，如果想更一般地得到 $N(\mu,\sigma^2)$ 的随机数可以由公式 $y_i = \mu + \sigma x_i$ 计算出来。

- **Poisson 分布随机数**

在 MATLAB 的统计学工具箱中，提供了大批生成其他分布随机数的函数，如生成 Poisson 分布的函数 poissrnd()，该函数的调用格式为：

```
x=poissrnd(lam,n)
```

其中 n 为随机数向量的个数，lam 为 Poisson 分布的参数 λ，假设 Poisson 分布的概率密度为 $P(n) = \lambda^n \mathrm{e}^{-\lambda}/n!$。

概率密度：用 hist() 函数可以将一个数据向量按其大小分配到各个格子里，并求出每个格子内分配的个数，所以该函数可以求取数据向量的概率密度。假设可以均匀地设置各个格子的宽度，则可以通过下面的语句求出：

```
y=hist(x,xx)
```

其中 x 为给定的数据向量，xx 为选定的格子所构造出的向量，则 y 可以算出各个格子内所分配的数据个数，这样其相应的概率密度就可以由 y/(length(x)*dx) 近似得出，其中 dx 为格子的宽度。

【例 3.55】 用 randn() 函数生成一组正态分布随机数，然后求出其概率密度，并和理论的概率密度进行比较。假设可以生成 30000 个随机数，并在 $[-3,3]$ 区间内设置 30 个格子，这样可以用下面的语句计算出生成数据的概率密度的近似值

```
>> x=randn(30000,1); xx=linspace(-3,3,30);
   y=hist(x,xx); yp=y/(length(x)*(xx(2)-xx(1)));
```

这时得出的概率密度由 yp 返回。事实上，标准正态分布的概率密度为 $p(e)=e^{-x^2/2}/\sqrt{2\pi}$，故可以用下面语句求出理论概率密度，并和得出的近似值相比较，结果如图 3-21 所示。

```
>> p0=exp(-xx.^2/2)/sqrt(2*pi);
   bar(xx,yp); hold on; plot(xx,p0)
```

可见，由 MATLAB 生成的伪随机数的概率密度与理论值匹配是较理想的。在统计学工具箱中还提供了多个在理论上求概率密度的函数，如正态分布的概率密度可以由 p=normpdf(xx,μ,σ) 求出。

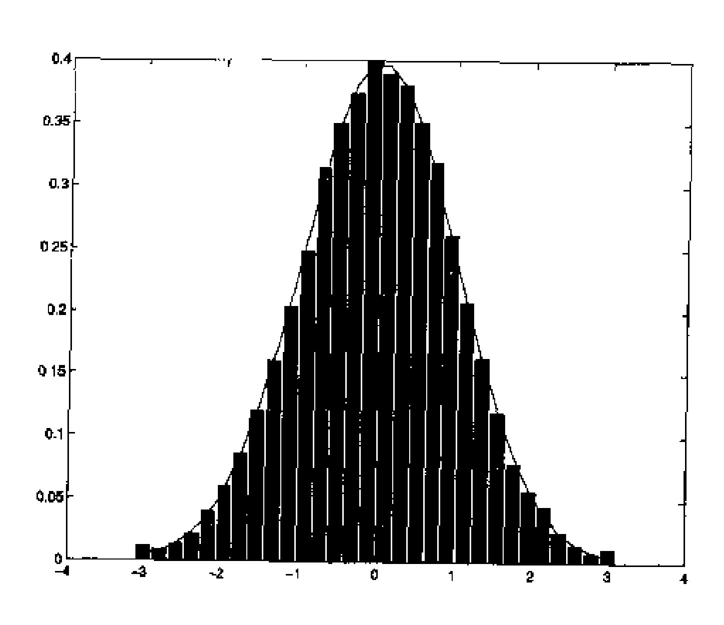


图 3-21 正态分布数据的概率密度比较

实验数据或随机数据也可以由 MATLAB 给出的数据处理函数容易地进行，比如函数 mean() 可以立即求出向量的均值；函数 std() 可以求出向量的标准方差；函数 median() 可以求出向量的中间值；函数 cov() 可以对矩阵进行处理，得出数据的协方差。

- **数据的相关分析**

假设在实验中测出两组数据，$x_i, y_i, (i=1,2,\cdots,n)$，则可以由下面的式子计算出两组数据的相关系数

$$r=\frac{\sqrt{\sum(x_i-\bar{x})(y_i-\bar{y})}}{\sqrt{\sum(x_i-\bar{x})}\sqrt{\sum(y_i-\bar{y})}} \tag{3.106}$$

MATLAB 提供了corrcoef() 函数，可以求出数据的相关系数矩阵。事实上，相关系数矩阵实际上是协方差矩阵元素按列进行归一化的结果。

〖例 3.56〗 利用 MATLAB 提供的函数可以先生成满足正态分布的 10000 × 5 伪随机数矩阵，然后可以用 mean() 和 std() 函数求出各列元素的均值和标准方差，再用 corrcoef() 函数求出这 5 列随机数据的相关系数矩阵。

```
>> S=randn(10000,5); M=mean(S), D=std(S)
M =
    0.0011    0.0066    0.0009    0.0264    0.0101
D =
    1.0011    1.0036    1.0049    1.0058    1.0061
>> V=corrcoef(S)
V =
    1.0000    0.0119    0.0051   -0.0114   -0.0011
    0.0119    1.0000    0.0093   -0.0012    0.0071
    0.0051    0.0093    1.0000    0.0048    0.0095
   -0.0114   -0.0012    0.0048    1.0000   -0.0017
   -0.0011    0.0071    0.0095   -0.0017    1.0000
```

由满足标准正态分布的随机数性质可以显然地看出，上面的结果是正确的，如产生的均值都很小，标准方差接近于 1 等。此外，由于其相关系数矩阵趋于单位矩阵，故由 randn() 生成的伪随机数据是独立的。

对这些离散点可以由下面的式子定义 x_i 序列的自相关函数

$$c_{xx}(k)=\frac{1}{N}\sum_{l=1}^{n-[k]-1}x(l)x(k+l),\ \ 0\le k\le m-1 \tag{3.107}$$

其中 $m<n$。类似地，还可以定义出互相关函数

$$c_{xy}(k)=\frac{1}{N}\sum_{l=1}^{n-[k]-1}x(l)y(k+l).\ \ 0\le k\le m-1 \tag{3.108}$$

MATLAB 下提供了求取和绘制自相关函数和互相关函数的程序，分别为 autocorr() 和 crosscorr()，这两个函数的调用格式分别为：

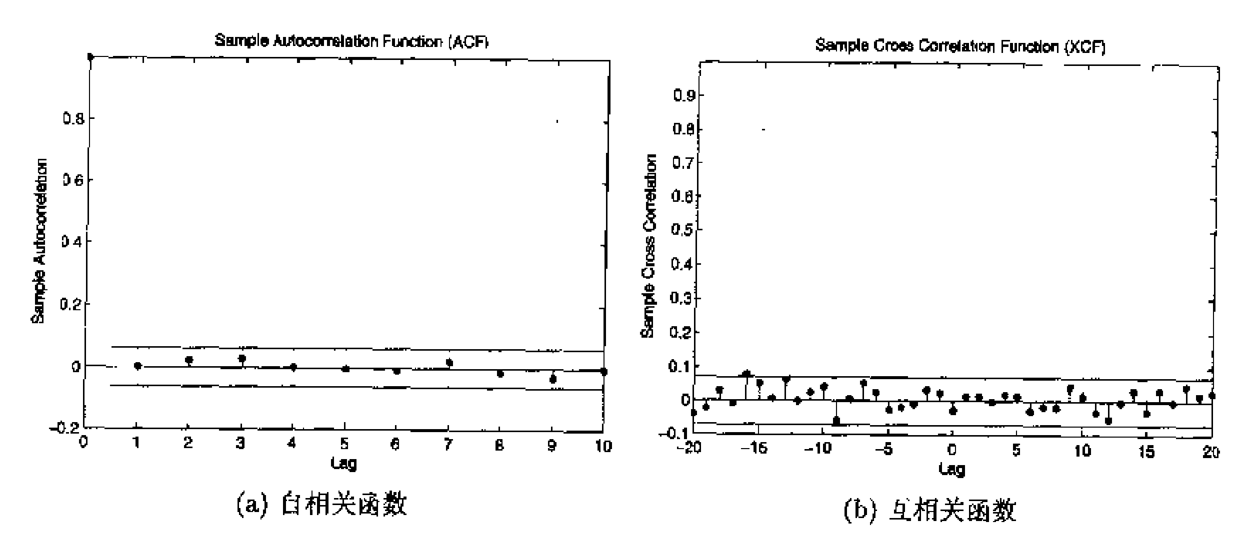
```
[Cxx, nLags, Bounds] = autocorr(x, nLag, nSTD);
[Cxy, nLags, Bounds] = crosscorr(x, y, nLag, nSTD);
```

其中 x, y 为数据向量，nLags 为公式中 m 的值 (默认为 20)，nSTD 为和标准方差有关的数值，它取默认值 2 是大约等于 95% 的置信度。返回的 Cxx 和 Cxy 分别为自相关函数和互相关函数的结果，如果不返回任何变量，则将自动绘制出带有置信区域的相关函数图形。

〖例 3.57〗 考察 MATLAB 产生的随机数的相关性，可以给出如下的语句

```
>> x=randn(1000,1);  % 产生 1000 个正态发布随机数
   y=randn(800,1); % 产生 800 个数，虽然函数一致，但种子不同，数据也不同
   autocorr(x,10); % 自动绘制自相关函数曲线，带 95% 置信度
   figure; crosscorr(x,y);  % 绘制互相关函数曲线，
   set(gca,'ylim',[-0.1,1]); % 纵轴采用同一尺度，以便比较
```

由这些语句得出的自相关函数和互相关函数曲线分别如图 3-22 (a) 和 (b) 所示。可以看出，这样生成的数据从相关函数角度看还是令人满意的。



(a) 自相关函数　　(b) 互相关函数

图 3-22　随机数据的相关函数分析

- **功率谱密度估计**

由给定的离散数据向量 y，在 MATLAB 中给出了函数 psd() 来求取其功率谱密度，但实际使用发现，该函数并非很令人满意，故这里引入基于 Welch 变换的估计算法[39]:

假设 n 为数据序列 y 的长度，则可以将这些点可以分为 m 长的 $K=[n/m]$ 个段落

$$x^{(i)}(k)=y[k+(i-1)m],\ 0<k\leqslant m-1,\ 1\leqslant i\leqslant K \tag{3.109}$$

用 Welch 算法，可以得出下面 K 个式子

$$J_m^{(i)}(\omega)=\frac{1}{mU}\left|\sum_{k=0}^{m-1}x^{(i)}(k)w(k)\mathrm{e}^{-\mathrm{j}\omega k}\right|^2 \tag{3.110}$$

其中 $w(k)$ 为数据处理窗口，例如它可以取作 Hamming 窗口

$$w(k)=0.54-0.46\cos\left(\frac{2\pi k}{m-1}\right),\quad k=0,\cdots,m-1 \tag{3.111}$$

且

$$U=\frac{1}{m}\sum_{k=0}^{m-1}w^2(k) \tag{3.112}$$

这样可以最终求出该信号的功率谱密度估计

$$P_{xx}^{w}(\omega)=\frac{1}{K}\sum_{j=1}^{K}J_{m}^{(i)}(\omega) \tag{3.113}$$

下面给出其具体实现步骤[50]：

- 用 fft() 函数对每个数据段计算 $X_m^{(i)}(l)=\sum_{k=0}^{m-1}x^{(i)}(k)w(k)\mathrm{e}^{-\mathrm{j}[2\pi/(m\Delta t)]lk}$。
- 对每个段落计算 $\left|X_m^{(i)}(l)\right|^2$，并计算累加和 $Y(l)=\sum_{i=1}^{K}\left|X_m^{(i)}(l)\right|^2$。
- 由下式求出数据的功率谱密度

$$P_{xx}^{w}\left(\frac{2\pi}{m\Delta t}l\right)=\frac{1}{KmU}Y(l) \tag{3.114}$$

由于使用了 fft() 函数来计算 $X_m^{(i)}$，所以有下面两点应该注意：

- MATLAB 提供的 fft() 函数用于连续 FFT 计算，而这里需要的是离散 Fourier变换，所以得出的 P_{xx}^{w} 应该再乘以 Δt。
- 为了使本算法计算效率最高，则 m 的值应该取 2 的整数次方。

依照前面的算法，可以编写出 psd_estm() 来估计给出序列的功率谱密度，该函数的清单如下：

```
function [Pxx,f] =psd_estm(y,m,T)
k=[0:m-1]; Y=zeros(1,m);
m2=floor(m/2);
f=k(1:m2)*2*pi/(length(k)*T);
w=0.54-0.46*cos(2*pi*k/(m-1));
K=floor(length(y)/m);
U=sum(w.^2)/m;
for i=1:K
   xi=y((i-1)*m+k+1)';
   Xi=fft(xi.*w);
   Y=Y+abs(Xi).^2;
end
Pxx=Y(1:m2)*T/(K*m*U);
```

在上面给出的程序中，为了避免假频现象，只取变换的一半信息。在该函数中，y、m 和定义一致，T 为数据的采样周期 Δt，计算后返回的 f 和 Pxx 分别为频率和功率谱密度。

3.7 习 题

(1) 感受 MATLAB 在求解逆矩阵上的运算效率。想求一个 n 阶随机矩阵的逆，分别取 $n=550$ 和 $n=1550$，测试矩阵求逆所需的时间及结果的正确性。

(2) 对下面给出的各个矩阵求取各种参数，如矩阵的行列式、迹、秩、特征多项式、范数等。

$$\boldsymbol{A}=\begin{bmatrix}7.5&3.5&0&0\\8&33&4.1&0\\0&9&103&-1.5\\0&0&3.7&19.3\end{bmatrix},\ \boldsymbol{B}=\begin{bmatrix}5&7&6&5\\7&10&8&7\\6&8&10&9\\5&7&9&10\end{bmatrix}$$

$$\boldsymbol{C}=\begin{bmatrix}1&2&3&4\\5&6&7&8\\9&10&11&12\\13&14&15&16\end{bmatrix},\ \boldsymbol{D}=\begin{bmatrix}3&-3&-2&4\\5&-5&1&8\\11&8&5&-7\\5&-1&-3&-1\end{bmatrix}$$

(3) 若 $\boldsymbol{V}=[1,2,3,4,5]$，构造一个 Vandermonde 矩阵，求出其特征多项式并验证 Hamilton-Cailey 定理，定量分析误差的大小。如果用 `poly1()` 函数取代 `poly()` 函数是否能改善精度？

(4) 对上面的各个矩阵进行三角分解和奇异值分解、求取特征值与特征向量。对上面的对称矩阵 $\boldsymbol{B}$ 作 Cholesky 分解，并验证所得出的结果。

(5) 求解下面的线性代数方程

$$\text{(a)}\ \begin{bmatrix}7&2&1&-2\\9&15&3&-2\\-2&-2&11&5\\1&3&2&13\end{bmatrix}\boldsymbol{X}=\begin{bmatrix}4\\7\\-1\\0\end{bmatrix},\quad \text{(b)}\ \begin{bmatrix}1&3&2&13\\7&2&1&-2\\9&15&3&-2\\-2&-2&11&5\end{bmatrix}\boldsymbol{X}=\begin{bmatrix}9&0\\6&4\\11&7\\-2&-1\end{bmatrix}$$

并验证得出的解真正满足原方程。

(6) 对下面给出的复数矩阵进行分析，分别求出它的矩阵参数、逆矩阵、秩、特征值与特征向量，并验证 Hamilton-Cailey 定理。

$$\boldsymbol{A}=\begin{bmatrix}0.2368&0.2471&0.2568&1.2671\\1.1161&0.1254&0.1397&0.1490\\0.1582&1.1675&0.1768&0.1871\\0.1968&0.2071&0.2168&0.2271\end{bmatrix}+\begin{bmatrix}0.1345&0.1768&0.1852&1.1161\\1.2671&0.2017&0.7024&0.2721\\-0.2836&-1.1967&0.3558&-0.2078\\0.3536&-1.2345&2.1185&0.4773\end{bmatrix}\mathrm{i}$$

(7) 求出下面给出的矩阵的秩和 Moore-Penrose 广义逆矩阵，并验证它们是否满足 Moore-Penrose 逆矩阵的条件。

$$\boldsymbol{A}=\begin{bmatrix}2&2&3&1\\2&2&3&1\\4&4&6&2\\1&1&1&1\\-1&-1&-1&3\end{bmatrix},\ \boldsymbol{B}=\begin{bmatrix}4&1&2&0\\1&1&5&15\\3&1&3&5\end{bmatrix}$$

(8) 对余弦函数、反正弦函数及对数函数分别编写求矩阵变换的MATLAB 函数，并利用编写的反正弦程序段对例子中的正弦结果进行检验，并计算例 3.16 中给出的矩阵。这里涉及的各个函数的幂级数展开公式如下：

(a) $\cos \boldsymbol{A}=\boldsymbol{I}-\dfrac{1}{2!}\boldsymbol{A}^2+\dfrac{1}{4!}\boldsymbol{A}^4-\dfrac{1}{6!}\boldsymbol{A}^6+\cdots+\dfrac{(-1)^n}{(2n)!}\boldsymbol{A}^{2n}+\cdots$

(b) $\arcsin \boldsymbol{A}=\boldsymbol{A}+\dfrac{1}{2\cdot 3}\boldsymbol{A}^3+\dfrac{1\cdot 3}{2\cdot 4\cdot 5}\boldsymbol{A}^5+\dfrac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6\cdot 7}\boldsymbol{A}^7+\cdots+\dfrac{(2n)!}{2^{2n}(n!)^2(2n+1)}\boldsymbol{A}^{2n+1}+\cdots$

(c) $\ln \boldsymbol{A}=\boldsymbol{A}-\boldsymbol{I}-\dfrac{1}{2}(\boldsymbol{A}-\boldsymbol{I})^2+\dfrac{1}{3}(\boldsymbol{A}-\boldsymbol{I})^3-\dfrac{1}{4}(\boldsymbol{A}-\boldsymbol{I})^4+\cdots+\dfrac{(-1)^{n+1}}{n}(\boldsymbol{A}-\boldsymbol{I})^n+\cdots$

(9) 给定下面特殊矩阵 $\boldsymbol{A}$，试利用符号运算工具箱求出其逆矩阵、特征值，并求出状态转移矩阵 $\mathrm{e}^{\boldsymbol{A}t}$ 的解析解。

$$\boldsymbol{A}=\begin{bmatrix} -9 & 11 & -21 & 63 & -252 \\ 70 & -69 & 141 & -421 & 1684 \\ -575 & 575 & -1149 & 3451 & -13801 \\ 3891 & -3891 & 7782 & -23345 & 93365 \\ 1024 & -1024 & 2048 & -6144 & 24572 \end{bmatrix}$$

(10) 试用 MATLAB 的符号运算工具箱直接求解下面的微积分问题：

(a) 不定积分 $\displaystyle\int \frac{x^3+3x^2-5}{(x^2-2x-6)(x^3+x+1)}\mathrm{d}x$。

(b) 对上面的结果进行微分，看是否能还原原函数。

(c) 对 $x^2\sin(\cos x^2)\cos x$ 函数作 20 项 Taylor 幂级数展开。

(d) $\displaystyle\lim_{n\to\infty}\left(1+\frac{1}{2}+\cdots\frac{1}{n}-\ln n\right)$。

(11) 改造 lorenzeq() 函数，使之以 β,σ,ρ 为附加参数，这样这些数据可以在 MATLAB 的工作空间中给定。绘制不同参数下的相空间轨迹，并弄清楚微分方程求解函数中函数句柄和函数名引用之间的区别。

(12) 考虑著名的 Rössol 化学反应方程组

$$\begin{cases} \dot{x}=y+z \\ \dot{y}=x+ay \\ \dot{z}=b+(x-c)z \end{cases}$$

选定 $a=b=0.2, c=5.7$，绘制仿真结果的三维相轨迹，并得出其在 x-y 平面上的投影。

(13) 考虑下面的化学反应系统的反应速度方程组[20]

$$\begin{cases} \dot{y}_1=-0.04y_1+10^4y_2y_3 \\ \dot{y}_2=0.04y_1-10^4y_2y_3-3\times 10^7y_2^2 \\ \dot{y}_3=3\times 10^7y_2^2 \end{cases}$$

其初值为 $y_1(0)=1, y_2(0)=y_3(0)=0$。该方程往往被认为是刚性方程，试就此模型分析当采用 ode45() 是否能正确求解，如果不能求解应该如何解决问题？

(14) 考虑下面给出的单位质量、单位长度的单摆方程

$$\begin{cases} \ddot{x} = -\lambda x \\ \ddot{y} = -\lambda y - \mathrm{g} \\ x^2 + y^2 - 1 = 0 \end{cases}$$

其中 g=9.81 为重力加速度。假设 $\lambda = 0.5$，且给定系统的初值为 $x(0) = 0.5, y(0) = \sqrt{1 - 0.5^2}$，$\dot{x}(0) = \dot{y}(0) = 0$，试求解相应的微分代数方程。

(15) 假设已知微分方程组

$$\begin{cases} \ddot{x}\sin\dot{y} + \ddot{y}^2 = -2xy + x\ddot{x}\dot{y} \\ x\ddot{x}\ddot{y} + \cos\ddot{y} = 3y\dot{x} \end{cases}$$

试选择一组状态变量，将该方程转换成一阶常微分方程组。

(16) 考虑简单的线性微分方程

$$y^{(4)} + 3y^{(3)} + 3\ddot{y} + 4\dot{y} + 5y = \mathrm{e}^{-3t} + \mathrm{e}^{-5t}\sin(4t + \pi/3)$$

且方程的初值为 $y(0) = 1, \dot{y}(0) = \ddot{y}(0) = 1/2, y^{(3)}(0) = 0.2$，试求该方程的解析解和数值解，并比较二者得出的曲线。

(17) 考虑 Van der Pol 方程 $\ddot{y} + \mu(y^2 - 1)\dot{y} + y = 0$，试求解 $\mu = 1$，且边值 $y(0) = 1, y(5) = 3$ 时方程的数值解。如果假设 μ 为自由参数，试求出满足边值条件，且满足 $\dot{y}(5) = -2$ 时方程的数值解及 μ 的值，并绘图验证之。

(18) 求解下面的最优化问题。

(a) $\min\limits_{\boldsymbol{x}\ \text{s.t.}\begin{cases} 4x_1^2 + x_2^2 \leq 4 \\ x_1, x_2 \geq 0 \end{cases}} \left(x_1^2 - 2x_1 + x_2\right)$，(b) $\max\limits_{\boldsymbol{x}\ \text{s.t.}\ x_1 + x_2 + 5 = 0} \left[(x_1 - 1)^2 - (x_2 - 1)^2\right]$

(c) $\min\limits_{\boldsymbol{x}\ \text{s.t.}\begin{cases} x_1/4 - 60x_2 - x_3/25 + 9x_4 \leq 0 \\ x_1/2 - 90x_2 - x_3/50 + 3x_4 \leq 0 \\ x_3 \leq 1 \\ x_1, x_2, x_3, x_4 \geq 0 \end{cases}} \left(-\dfrac{3}{4}x_1 + 150x_2 - \dfrac{1}{50}x_3 + 6x_4\right)$

其中 (b) 是二元函数的最优化问题，试用图示的方法判定得出的结果是否合理，并解释之。

(19) 假设有一组实测数据

x_i	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y_i	2.3201	2.6470	2.9707	3.2885	3.6008	3.9090	4.2147	4.5191	4.8232	5.1275

(a) 绘制出各种插值算法下的拟合效果。

(b) 假设已知该数据可能满足的原型函数为 $y(x) = ax + bx^2\mathrm{e}^{-cx} + d$，试求出满足下面数据的最小二乘解 a, b, c, d 的值。

(20) 生成一个随机数序列，将该随机数序列循环左移 10 位，就可以构成另一个随机数序列，求这两个随机数序列的互相关函数。

提示：假设原序列 x_i 长度为 n，则新序列可以由下面的公式生成，但不要用循环结构生成该序列

$y_i = x_{i+10}$, $(i = 1, \cdots, n-10)$，且 $y_{n-10+i} = x_i$, $(i = 1, \cdots, 10)$。

(21) 观察 MATLAB 的随机数发生函数 randn()，检验在不同的随机数种子下发生数据的相关函数和功率谱密度是否有明显的变化。

第 4 章 MATLAB/Simulink 下数学模型建立与仿真

Simulink 是 The MathWorks 公司于 1990 年推出的产品，是用于 MATLAB 下建立系统框图和仿真的环境。该环境刚推出时的名字叫 Simulab，由于其名字很类似于当时的一个很著名的语言—— Simula 语言，所以次年更名为 Simulink。从名字上看，立即就能看出该程序有两层含义，首先，“Simu”一词表明它可以用于计算机仿真，而 “Link” 一词表明它能进行系统连接，即把一系列模块连接起来，构成复杂的系统模型。正是由于它的这两大功能和特色，使得它成为仿真领域首选的计算机环境。

早在 Simulink 出现之前，仿真一个给定框图的连续系统是件很复杂的事，当时 MATLAB 虽然已经支持较简单的常微分方程求解，但用语句的方式建立起整个系统的状态方程模型还是比较困难的事，所以需要借助于其他的仿真语言工具，如 ACSL 语言[34]，来描述系统模型，并对之进行仿真。当时采用这样的语言建立模型需要很多的手工编程，很不直观，对复杂的问题来说出错是难以避免的，结果经常难以令人相信；另外，由于过多的手工编程，使得解决问题的时间浪费很多，很不经济；最致命的，因为它们毕竟属于不同的语言，相互之间传送数据很不方便，这很大程度上限制了 ACSL 和 MATLAB 语言的联合使用。所以从 Simulink 一出现起，很多惯用 ACSL 的用户纷纷弃用该语言，改用 Simulink 作为主要的仿真工具。

在 Simulink 类软件出现之前，为了考核各类控制系统 CAD 软件的建模难易程度、算法的精度等指标，本领域有影响的专家提出了一些测试基准问题 (benchmark problems)，由于 Simulink 的出现，使得原来的基准问题能够轻而易举地解决了。

本章首先在第 4.1 节中系统地概述 Simulink 中各个模块库，并概略介绍其中一些常用的模块；在第 4.2 节中将介绍模块的使用方法，如模块旋转翻转、模块连接及参数修改，并将介绍搭建起来的 Simulink 模型的仿真方法；第 4.3 节将通过一些有代表性的例子演示 Simulink 在模型表示和仿真中的应用；第 4.4 节中将介绍一类特殊的系统模型——线性系统模型的时域、频域分析和 Simulink 在线性系统仿真中的应用；第 4.5 节中将介绍连续系统在随机输入作用下的仿真算法和仿真结果的统计分析技术，在该节中将分析系统的概率密度、相关函数和功率谱密度的理论值和基于仿真结果的估算；在第 4.6 节中将介绍分形系统的仿真方法及其 MATLAB 语言实现，着重介绍分形树、 Julia 集和 Mandelbrot 集的仿真与图形表示方法。

4.1 Simulink 模块库简介

在 MATLAB 命令窗口下给出 `simulink` 命令，或单击 MATLAB 工具栏中的 Simulink 图标，则将打开 Simulink 模型库窗口，如图 4-1 所示。

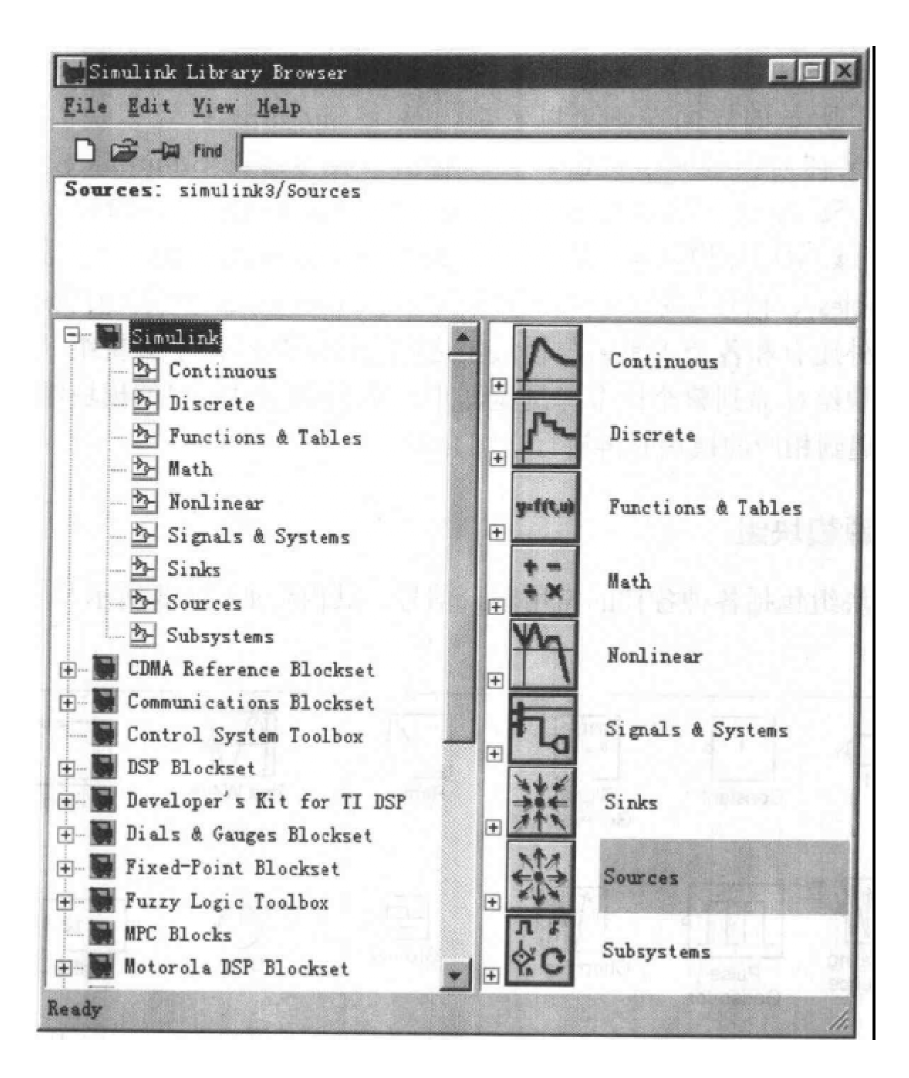


图 4-1 Simulink 模块库界面

熟悉早期版本的还可以在 MATLAB 下键入 **simulink3** 命令来打开整个模块库，这时模块库的表现形式如图 4-2 所示，其表现形式和早期版本完全一致，从这个模块库直接

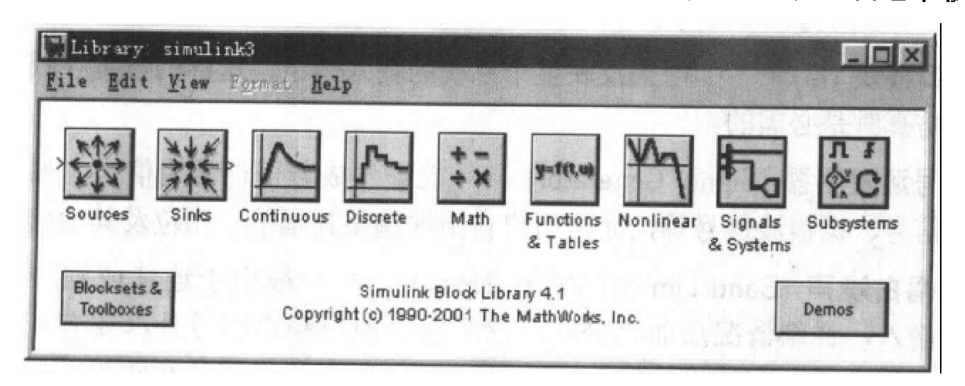


图 4-2 Simulink 模块库的其他显示形式

访问子模块库的方式也完全一致，不过作者还是建议使用新的界面去访问各个 Simulink 模块库。这里为了更好地介绍各个模块组的基本内容，我们还是采用传统的形式，这样

能更好地显示出每个模块组的全貌。

从如图 4-1 所示的界面左侧可以看出，整个 Simulink 模块库是由各个模块组构成，故该界面又称为模型库浏览器。可以看出，在标准的 Simulink 模块库中，包括信号源模块组 (Sources)、输出池模块组 (Sinks)、连续模块组 (Continuous)、离散模块组 (Discrete)、数学运算模块组 (Math)、非线性模块组 (Nonlinear)、函数与表格模块组 (Function & Tables)、信号与系统模块组 (Signals & Systems) 和子系统模块组 (Subsystems) 几个部分，此外还有和各个工具箱与模块集之间的联系构成的子模块组，用户还可以将自己编写的模块组挂靠到整个模型库浏览器下。本节中将对常用的模块组和模块作一个概述，在以后遇到相应的模块时再进行详细介绍。

4.1.1 信号源模块组

信号源模块组包括各种各样的常用输入信号，其内容如图 4-3 所示[①]。该模块组的主要模块为：

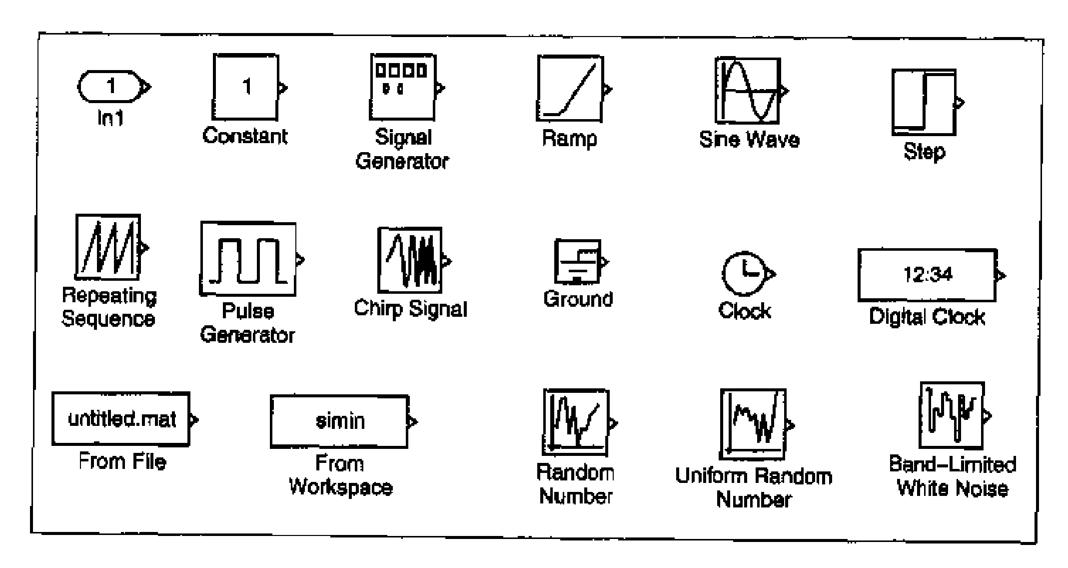


图 4-3 信号源模块组

- **输入端口模块** (In)[②] —— 用来反映整个系统的输入端子，这样的设置在模型线性化与命令行仿真时是必需的。
- **普通信号源发生器** (Signal Generator) —— 能够生成若干种常用信号，如方波信号、正弦波信号、锯齿波信号等，允许用户自由地调整其幅值、相位及其他信号。
- **带宽限幅白噪声** (Band-Limited White Noise) —— 一般用于连续或混杂系统的白噪声信号输入，详细情况后面将介绍。除了这样的白噪声信号外，还有一般随机数发生模块，如正态分布随机数模块 (Random number) 和均匀分布随机数模块 (Uniform Random Number) 等，但注意，这两个模块不能直接用于仿真连续系统。

[①]注意，为了排版方便，采用了图 4-2 中所示界面的模块组图形，并进行了模块的重新布置，使得每个模块的位置和整个组的排版篇幅较少，但内容应该和图 4-1 中所示的完全一致。

[②]在早期版本 (包括 MATLAB 6.0) 中，此模块应该位于信号与系统模块组中。

- **读文件模块** (From File) **和读工作空间模块** (From Workspace) —— 两个模块允许从文件或 MATLAB 工作空间中读取信号作为输入信号。
- **时间信号模块** (Clock) —— 生成当前仿真时钟，在与时间有关的指标求取中是很有意义的，例如获取系统的 ITAE 准则等，关于这方面的应用后面将详细介绍。
- **常数输入模块** (Constant) —— 此模块以常数作为输入，可以在很多模型中使用该模块。
- **接地线模块** (Ground) —— 一般用于表示零输入模块，如果一个模块的输入端子没有接任何其他模块，在 Simulink 仿真中经常给出错误信息，这样可以将该模块接入该输入端子即可避免错误信息。
- 各种其他类型的信号输入，如阶跃输入 (Step)、斜坡输入 (Ramp)、脉冲信号 (Pulse Generator)、正弦信号 (Sine Wave) 等，还允许由 Repeating Sequence 模块构造可重复的输入信号。

4.1.2 连续模块组

连续模块组包括常用的连续模块，连续模块组的内容如图 4-4 所示，包括：

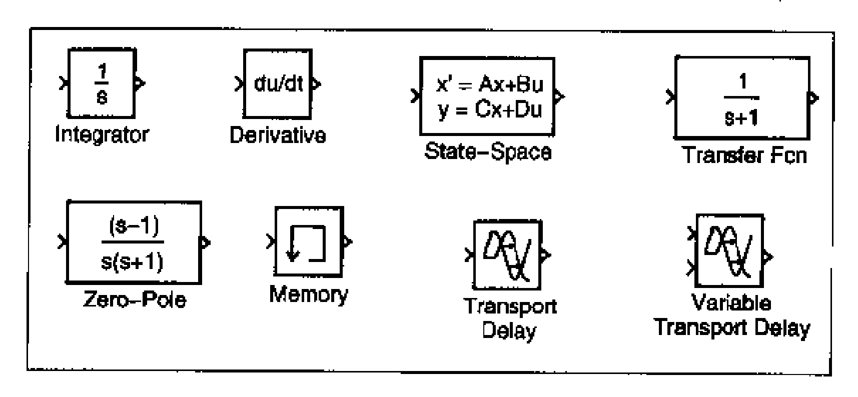


图 4-4　连续模块组

- **积分器** (Integrator) —— 连续动态系统最常用的元件，该模块将输入端信号经过数值积分，在输出端直接反映出来。在将常微分方程转换为图形表示时也必须使用此模块。积分器模块随着其采用不同的选项将有不同的变化形式，这将在后面的例子中进行介绍。
- **数值微分器** (Derivative) —— 该模块的作用是将其输入端的信号经过一阶数值微分，在输出端输出出来。在实际应用中应该尽量避免使用该模块。
- **线性系统的状态方程** (State-Space) —— 是线性系统的一种时域描述，系统的状态方程数学表示为

$$\begin{cases} \dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \\ \boldsymbol{y} = \boldsymbol{C}\boldsymbol{x} + \boldsymbol{D}\boldsymbol{u} \end{cases} \tag{4.1}$$

其中 $\boldsymbol{A}$ 矩阵是 $n \times n$ 方阵，$\boldsymbol{B}$ 为 $n \times p$ 矩阵，$\boldsymbol{C}$ 为 $q \times n$ 矩阵，$\boldsymbol{D}$ 为 $q \times p$ 矩阵，这又称为这些矩阵维数相容。在状态方程模块下，输入信号为 $\boldsymbol{u}$，而输出信号为 $\boldsymbol{y}$。

- **传递函数** (Transfer Fcn) —— 传递函数是频域下常用的描述线性微分方程的一种方法，通过引入 Laplace 变换可以将原来的线性微分方程在零初始条件下变换成“代数”的形式，从而以多项式的比值形式描述系统，传递函数的一般形式为

$$G(s)=\frac{b_1s^m+b_2s^{m-1}+\cdots+b_ms+b_{m+1}}{s^n+a_1s^{n-1}+a_2s^{n-2}+\cdots+a_{n-1}s+a_n} \tag{4.2}$$

  其中的分母多项式又称为系统的特征多项式，分母多项式的最高阶次又称为系统的阶次。物理可实现系统要满足 $m\le n$，这种情况下又称系统为正则 (proper) 的。传递函数实际上是输出的 Laplace 变换和输入的 Laplace 变换直接的比值。

- **零极点** (Pole-Zero) —— 将传递函数模型的分子和分母分别进行因式分解，则可以将其变换成

$$G(s)=K\frac{(s+z_1)(s+z_2)\cdots(s+z_m)}{(s+p_1)(s+p_2)\cdots(s+p_n)} \tag{4.3}$$

  其中 K 称为系统的增益，$-z_i,(i=1,\cdots,m)$ 称为系统的零点，而 $-p_i,(i=1,\cdots,n)$ 称为系统的极点。很显然，对实系数的传递函数模型来说，系统的零极点或者为实数，或者以共轭复数的形式出现。

- **时间延迟** (Transport Delay 或 Variable Transport Delay) —— 用于将输入信号延迟指定的时间后传输给输出信号。两个模块的区别在于：前者在模块内部参数中设置延迟时间，而后者将采用输入信号来定义延迟时间。

4.1.3 离散模块组

离散模块组主要用于建立离散采样系统的模型。离散模块组的内容如图 4-5 所示。该模块组主要包括：

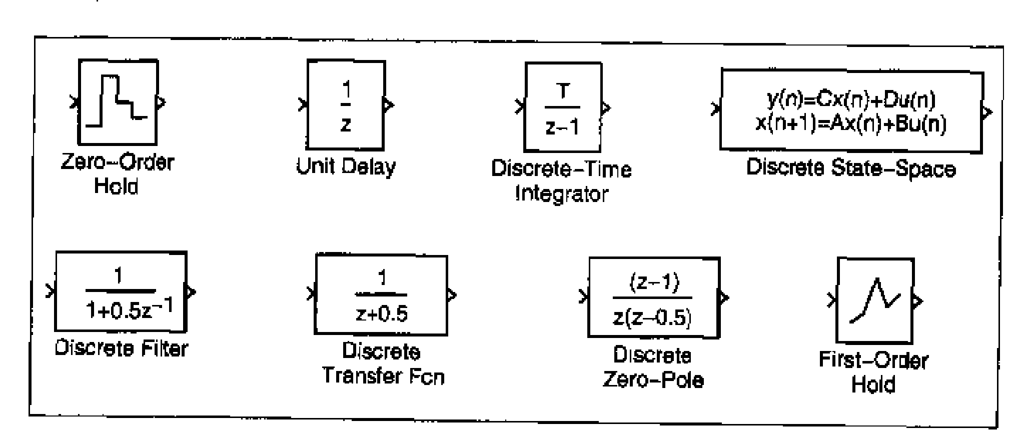


图 4-5 离散模块组

- **零阶保持器** (Zero-Order Hold) **和一阶保持器** (First-Order Hold) —— 前者在一个计算步长内将输出的值保持在同一个值上，而后者将依照一阶插值的方法计算一个计算步长下的输出值。

- **离散系统的传递函数和状态方程** —— 定义和连续系统的相类似，其定义分别为

$$\begin{cases} \boldsymbol{x}[(k+1)T] = \boldsymbol{A}\boldsymbol{x}(kT) + \boldsymbol{B}\boldsymbol{u}(kT) \\ \boldsymbol{y}(kT) = \boldsymbol{C}\boldsymbol{x}(kT) + \boldsymbol{D}\boldsymbol{u}(kT) \end{cases} \tag{4.4}$$

和

$$G(z) = \frac{b_1 z^m + b_2 z^{m-1} + \cdots + b_m z + b_{m+1}}{z^n + a_1 z^{n-1} + a_2 z^{n-2} + \cdots + a_{n-1} z + a_n} \tag{4.5}$$

其中 T 为采样系统的采样周期。模块组中的滤波器 (Filter)、单位延迟 (Unit Delay) 等都应该是离散系统传递函数的特殊情况。

4.1.4 函数与表格模块组

函数与表格模块组实现各种一维、二维或高维函数的查表，另外用户可以自己编写更复杂的函数，该模块组内容如图 4-6 所示。该模块组主要包含：

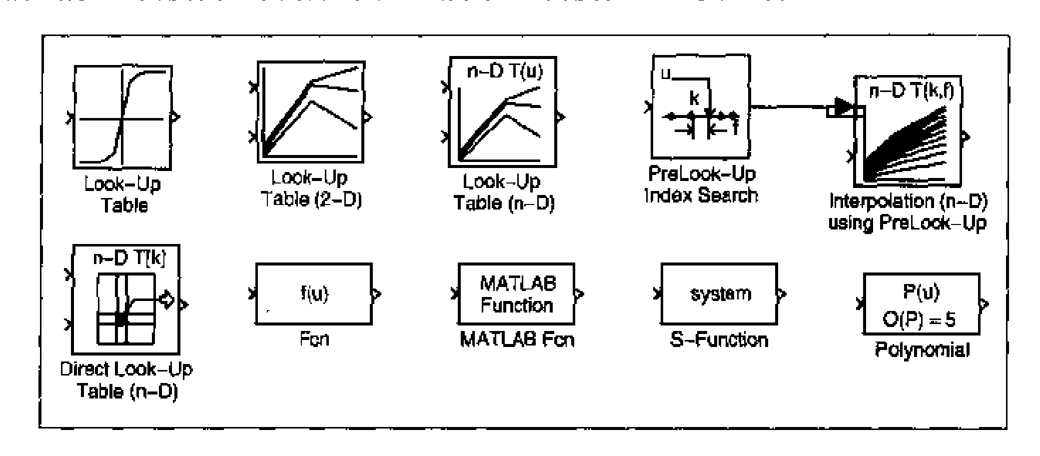


图 4-6 函数与查表模块组

- **一维查表模块** (Look-Up Table) —— 给出一组横坐标和纵坐标的参考值，则输入量经过查表和线性插值计算出输出值返回。
- **二维查表模块** (Look-Up Table 2D) —— 给出二维平面网格上的高度值，则输入的两个变量经过查表、插值运算，计算出模块的输出值。
- **函数计算模块** (Fcn) —— 可以将输入信号进行指定的函数运算，该模块可以对输入信号实现很复杂的函数运算，最后计算出模块的输出值。
- **MATLAB 函数的模块** (MATLAB Fcn) —— 可以将用户自己按照规定格式编写的 MATLAB 函数嵌入到 Simulink 模型中，这样就可以对输入进行运算，计算生成出输出信号。
- **S-函数模块** (S-Function) —— 按照 Simulink 规定的标准，允许用户编写自己的 S-函数，可以将 MATLAB 语句、C/C++ 语句、Fortran 语句或 Ada 语句等编写的函数在 Simulink 模块中执行，最后计算出模块的输出值。

4.1.5　数学运算模块组

数学函数模块组实现了各种各样的数学函数模块，如图 4-7 所示，该模块组主要的模块为：

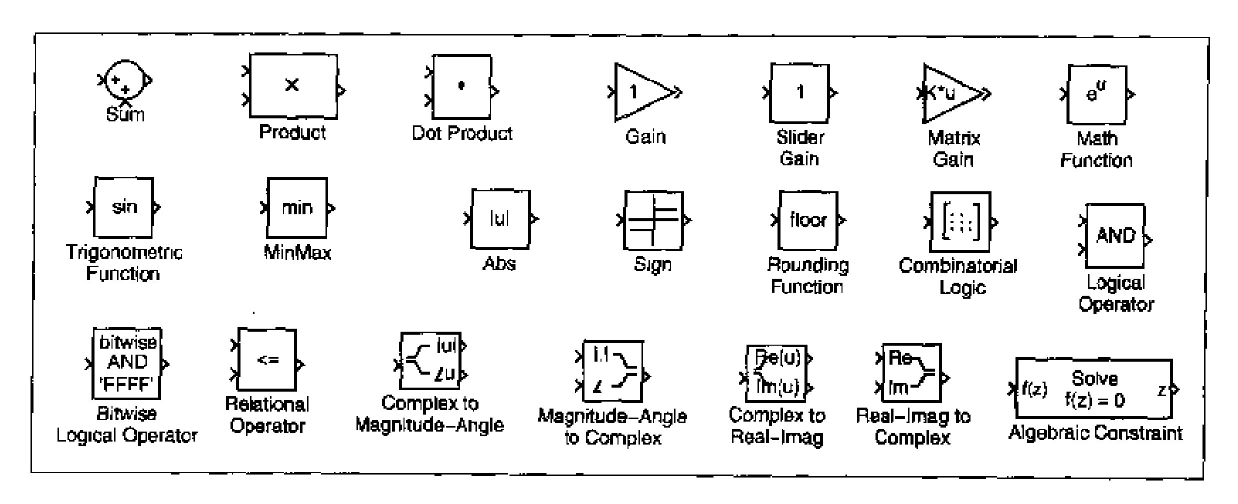


图 4-7　数学函数模块组

- **增益函数** (Gain) —— 输出信号等于输入信号乘以增益模块中指定的数值。更一般地，还有对矩阵进行乘法的矩阵增益模块 (Matrix Gain)。
- **求和模块** (Sum) —— 将输入的多路信号进行求和或求差，则可以计算出输出信号。在组建反馈控制系统框图时必须采用该模块。
- **代数约束模块** (Algebraic Constraint) —— 可以在 Simulink 模型中引入某些代数方程求解的算法，其功能是约束其输入信号，使其值为零，该模块可以用于微分代数方程的建模。
- **复数的实部虚部提取模块** (Complex to Real and Imag)、复数变换成幅值幅角的模块 (Complex to Magnitude-Angle) 及其反变换。
- 一般数学函数，如绝对值函数 (Abs)、符号函数 (Sign)、三角函数 (Trigonometric Function)、取整模块 (Rounding Function) 等。
- 数字逻辑模块，如逻辑运算模块 (Logic Operator)、组合逻辑模块 (Combinational Logic) 等，可以用这些模块容易地搭建起数字逻辑电路。

4.1.6　非线性模块组

非线性模块组包含一些常用的非线性运算模块，如图 4-8 所示。该模块组的主要模块包括：

- **Coulomb与黏性摩擦** (Coulomb & Viscous Friction)。
- **开关模块** (Switch 或 Multiport Switch) —— 由开关量的值选择出哪路输入信号直接产生输出信号，在很多场合下可以采用此模块。

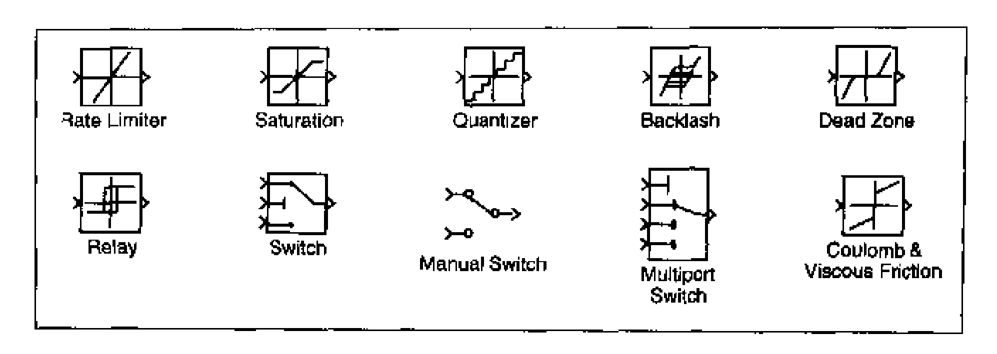


图 4-8 非线性模块组

- **磁滞回环模块** (Backlash)，和其在控制系统中的定义一致。
- 在此模块组中定义了很多分段线性的静态非线性模块，如死区非线性 (Dead Zone)、饱和非线性 (Saturation)、量化模块 (Quantizer)、继电模块 (Relay)、变化率限幅模块 (Rate Limiter) 等，其实其中很多模块可以由一维查表模块实现。

4.1.7 输出池模块组

输出池模块组中的模块实际上是包含那些能显示计算结果的模块，如图 4-9 所示。该模块组包括的模块为：

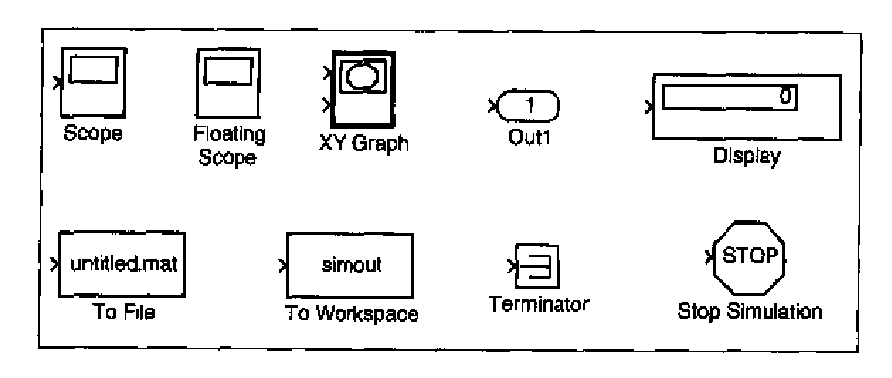


图 4-9 输出池模块组

- **输出端口模块** (Out)[①] —— 用来反映整个系统的输出端子，这样的设置在模型线性化与命令行仿真时是必需的，另外，系统直接仿真时这样的输出将自动在 MATLAB 工作空间中生成变量。
- **示波器模块** (Scope) —— 将输入信号在示波器中显示出来。
- **x-y 示波器** (x-y Graph) —— 将两路输入信号分别作为示波器的两个坐标轴，将信号的相轨迹显示出来。
- **工作空间写入模块** (To Workspace) —— 将输入信号直接写到 MATLAB 的工作空间中。该模块默认的写法是结构体型的数据，可以通过设置将之设置成矩阵型的。

[①]在早期版本 (包括 MATLAB 6.0) 中，此模块应该位于信号与系统模块组中。

- **写文件模块** (To File)　将输入的信号写到文件中。
- **数字显示模块** (Display) —— 将输入信号用数字的形式显示出来。
- **仿真终止模块** (Stop Simulation) —— 如果输入的信号为非零时，将强行终止正在进行的仿真过程。
- **信号终结模块** (Terminator) —— 可以将该模块连接到闲置的未连接的模块输出信号上，避免出现警告。

4.1.8　信号与系统模块组

信号与系统模块组包含的模块如图 4-10 所示。该模块组包含的主要内容为：

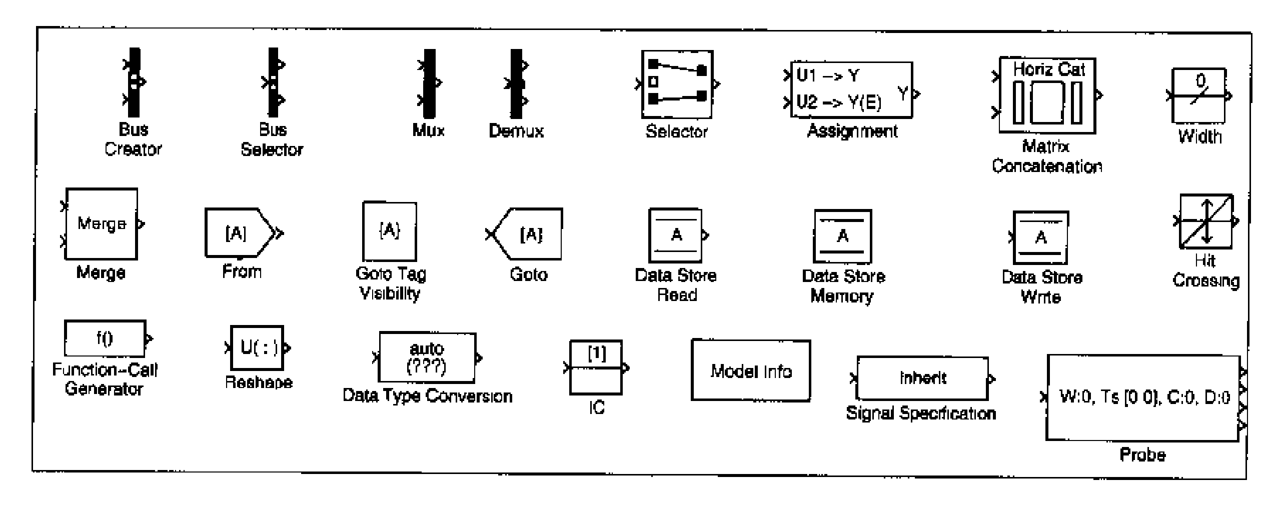


图 4-10　信号与系统模块组

- **混路器** (Mux) **和分路器** (Demux) —— 前者将多路信号依照向量的形式混合成一路信号。例如，可以将要观测的多路信号合并成一路，连接到示波器上显示，这样就可以将这些信号同时显示出来。分路器是将混路器组成的信号依照原来的构成方法解成多路信号。
- **模型信息显示模块** (Model Info) —— 允许显示模型的有关信息。
- **选路器** (Selector) —— 可以从多路输入信号中按希望的顺序输出所需路数的信号。
- 矩阵基本运算模块，如读矩阵模块 (From)，数据结构自动转换模块 (Data Type Conversion)，矩阵的重新定维模块 (Reshape) 等。

4.1.9　子系统模块组

子系统模块组包含了各种各样的子系统结构，如图 4-11 所示，其主要内容为：

- **空白子系统结构** (Atomic Subsystem) —— 搭建子系统模块，给出输入和输出端子，允许用户在其间绘制所需的子系统模型。

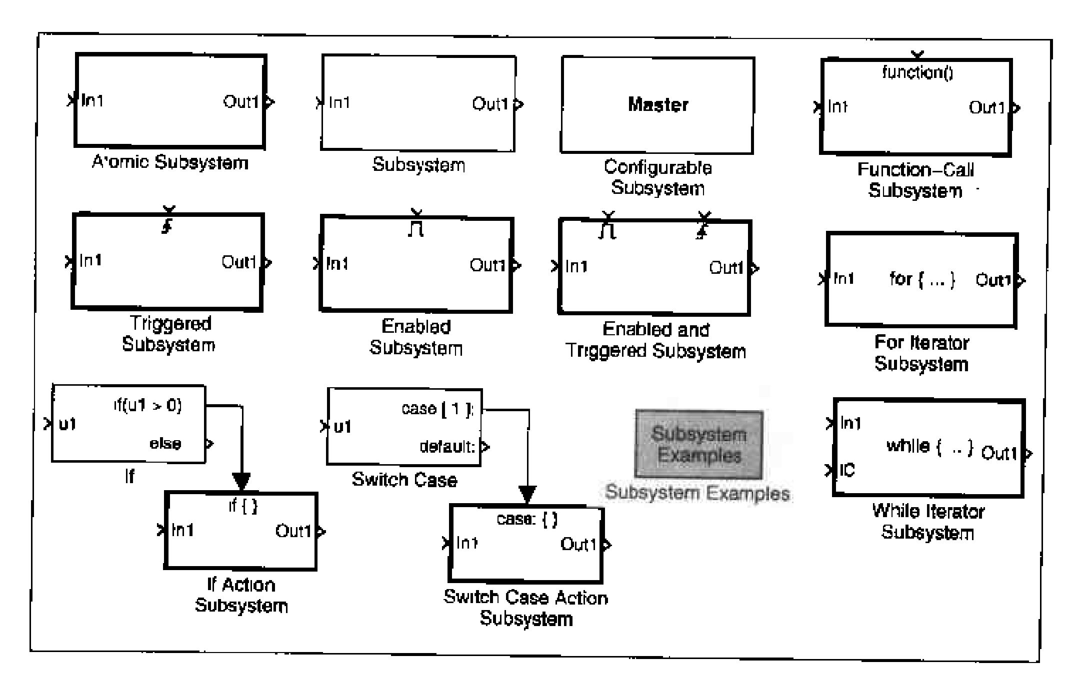


图 4-11　子系统模块组

- **触发子系统模块** (Triggered Subsystem) — 在触发信号发生时子系统可以工作，触发信号分为上升沿、下降沿等。
- **使能子系统** (Enabled Subsystem) — 在使能信号发生时子系统开始工作，用户可以自己构造使能信号。此外，还可以将使能信号和触发信号共同用于控制子系统的行为。
- **结构控制子系统** — 各种程序控制结构下子系统，包括 `for` 循环、`while` 循环等，还有转移语句 `if` 和开关 `switch` 模块。

4.1.10　其他模块组

事实上，一般构造系统的常用模块在这个基本模块库中都存在了，在其他模块集中包括了很多更具特色的模块，所以再配合齐全的模块集，在强大的 MATLAB 支持下，可以更方便、迅速、准确地解决系统仿真的问题。在很多模块集中通常只有一两个模块，但经常这些模块是作为和相应工具箱的接口。

这里只简单地介绍其中若干个常用的模块集。

- **通信系统仿真模块集** (Comm Blockset) **和 CDMA 模块集** (CDMA Blockset)：这两个模块集都是用于通信系统仿真的实用工具。
- **数字信号处理模块集** (DSP Blockset) 中定义了若干数字信号处理模块，如功率谱密度求取、自相关函数求取等，有关 DSP 模块集的详细内容后面将进一步介绍。
- **表盘模块集** (Dial and Gauges Blockset) 中提供了大量表盘类显示元件，通过 ActiveX 技术和 MATLAB 与 Simulink 环境进行数据交换的，可以将系统中的信号用表盘的形式进行显示，更类似于现场环境。

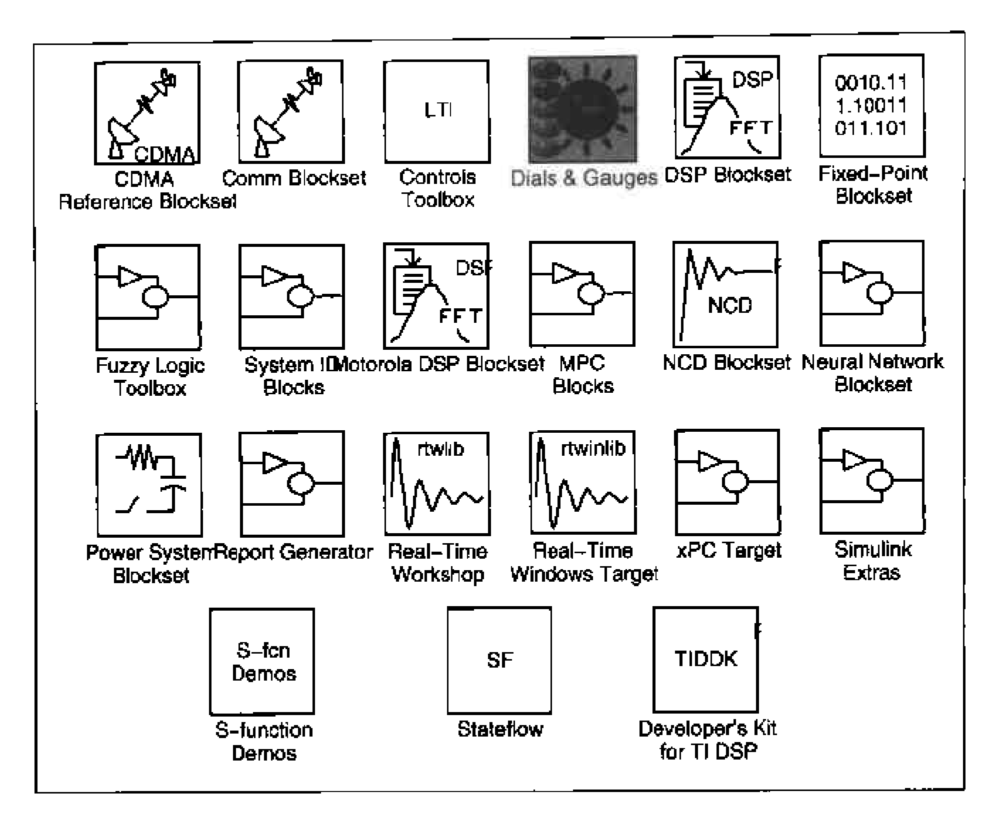


图 4-12　附加模块组库

- **非线性系统设计模块集** (Nonlinear Control Design Blockset，即 NCD Blockset) 基于数值优化算法提出了非线性系统最优控制设计的解决方案，有一定的特色。在后面将专门详细介绍该模块集。
- **一组和工具箱配套的模块集**，如控制系统模块集 (Control Blockset) 是为控制系统工具箱提供 Simulink 接口的，它支持线性模型对象在 Simulink 中直接应用，而无需再像连续模块组那样去赋底层的参数；模糊逻辑模块集 (Fuzzy Logic Blockset) 是为模糊逻辑工具箱提供接口的；神经网络工具箱 (Neural Network Blockset) 是为神经网络工具箱提供接口的，而模型预测控制工具箱 (Model Predictive Blockset) 是和预测控制工具箱提供接口的；系统辨识模块集 (System ID Blockset) 是为系统辨识工具箱建立接口的。还有其他大量的模块集，在此不一一列举。
- **电力系统模块集** (Power Systems Blockset) 给出的若干常用电子、电力与电机元件的 Simulink 模型，经过内部的变换可以转换成微分方程模型，在 Simulink 环境中进行仿真研究，本模块集将在后面详细介绍。
- **虚拟现实工具箱** (Virtual Reality Toolbox) 提供了虚拟现实设备的输入方法和三维视景显示方式，给用户带入虚拟的世界。
- **机构系统模块集** (SimMechanics Blockset®) 是 The MathWorks 公司 2001 年底正式推出的，它可以对各种机构系统进行仿真，并基于虚拟现实工具箱显示仿真结果。

- **实时控制类模块集**，如实时控制工具库 (Real-Time Workshop) 可以将 Simulink 模型翻译成 C 语言程序，加快执行速度；xPC 模块集可以进行半实物仿真等研究，定点控制模块集 (Fixed-Point Blockset) 仿照在微机控制中定字长、定点数据的处理方法进行仿真，对实时控制有一定的指导作用。

4.2 Simulink 模型的建立

4.2.1 模型窗口建立

在 Simulink 环境下，编辑模型的一般过程是：首先打开一个空白的编辑窗口，然后将模块库中模块复制到编辑窗口中，并依照给定的框图修改编辑窗口中模块的参数，再将各个模块按照给定的框图连接起来，这样就可以对整个模型进行仿真了。

在 Simulink 中打开一个空白的模型窗口有几种方法：

- 在 MATLAB 的命令窗口中选择 File | New | New Model 菜单项；
- 单击 Simulink 工具栏中的“新建模型”图标；
- 选中 Simulink 菜单系统中的 File | New | Model 菜单项；
- 还可以使用 new_system 命令来建立新模型，具体方法后面的章节中将详细介绍。

无论采用哪种方式，都将自动地打开一个如图4-13 所示的空白窗口模型，在该图上

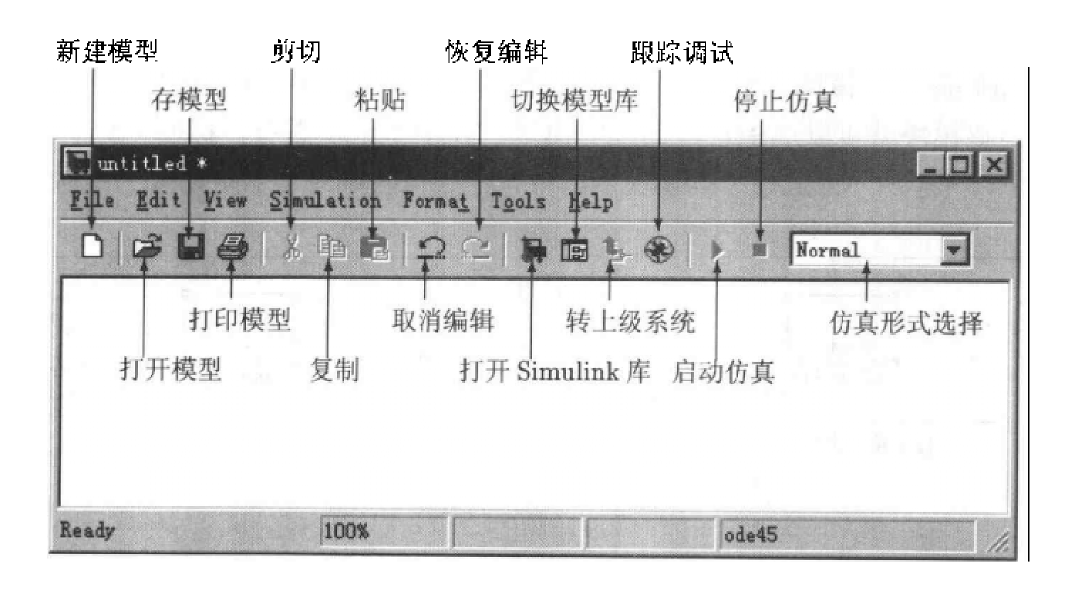


图 4-13 Simulink 模块编辑窗口

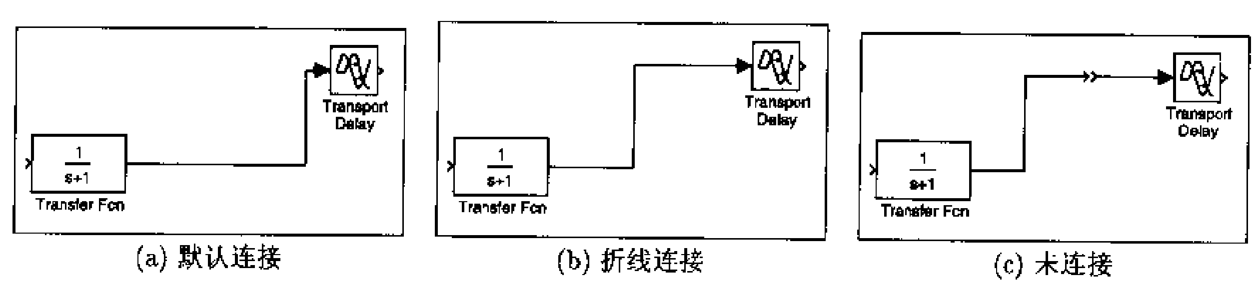
对该窗口工具栏中的各个工具按钮作了简要的描述。在这个窗口下我们就可以任意地编辑所需要的系统模型了。在后面各小节中将详细介绍模型的编辑、处理、仿真的方法。

4.2.2 模块的连接与简单处理

将两个模块连接起来在 Simulink 下连接起来是一件很简单的事，在每个允许输出的口都有一个输出的 > 符号表示，离开该模块，而输入端也有一个表示输入的 > 符号，进入

该模块。如果想连接起来两个模块，只需在前一个模块的输出口处按下鼠标左键，拖动鼠标至后一个模块的输入口处释放鼠标键，则 Simulink 会自动地将两个模块连接起来。如果想快速进行两个模块的连接，还可以先单击选中源模块，按下 Ctrl 键，再单击目标模块，这样将直接建立起两个模块的可靠连接。

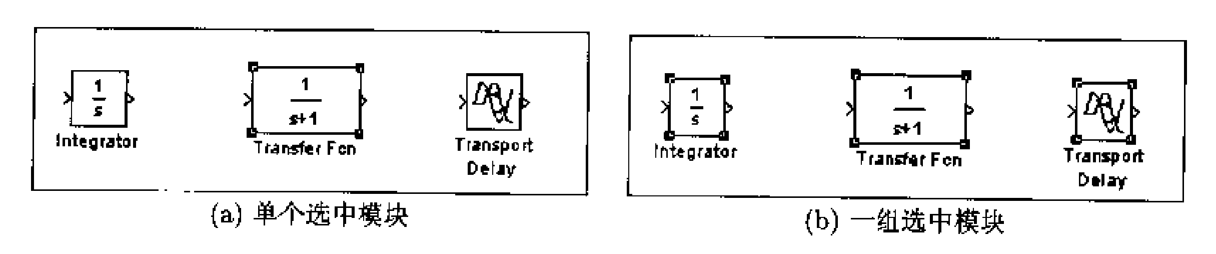
注意，正确连接之后，连线带有实心的箭头。如图 4-14 (a) 所示，如果在画图时，不一次性将前一个模块的输出和后一个模块的输入连接起来，而是先画一条水平线，在继续画垂直线，再画水平线，到后一个模块输入点处释放鼠标键，就可以得出如图 4-14 (b) 所示的连接效果。一般情况下，这样的连接方式更实用，因为用户可以自己控制模型的布线。如果连接不正确，则出现如图 4-14 (c) 所示的连线。



(a) 默认连接 (b) 折线连接 (c) 未连接

图 4-14 连接两个模块

有的时候，为了布线的美观和易读，经常需要将某个或某些模块进行旋转或翻转处理，在 Simulink 下对模块进行这样的处理是很容易的，首先应该选中该模块或模块组。用鼠标单击该模块就可以选中它，选中的模块的四个角出现黑点，标明它处于选中的状态，如图 4-15 (a) 所示。选择一些模块可以首先在选择区域的左下角处按下鼠标左键，然



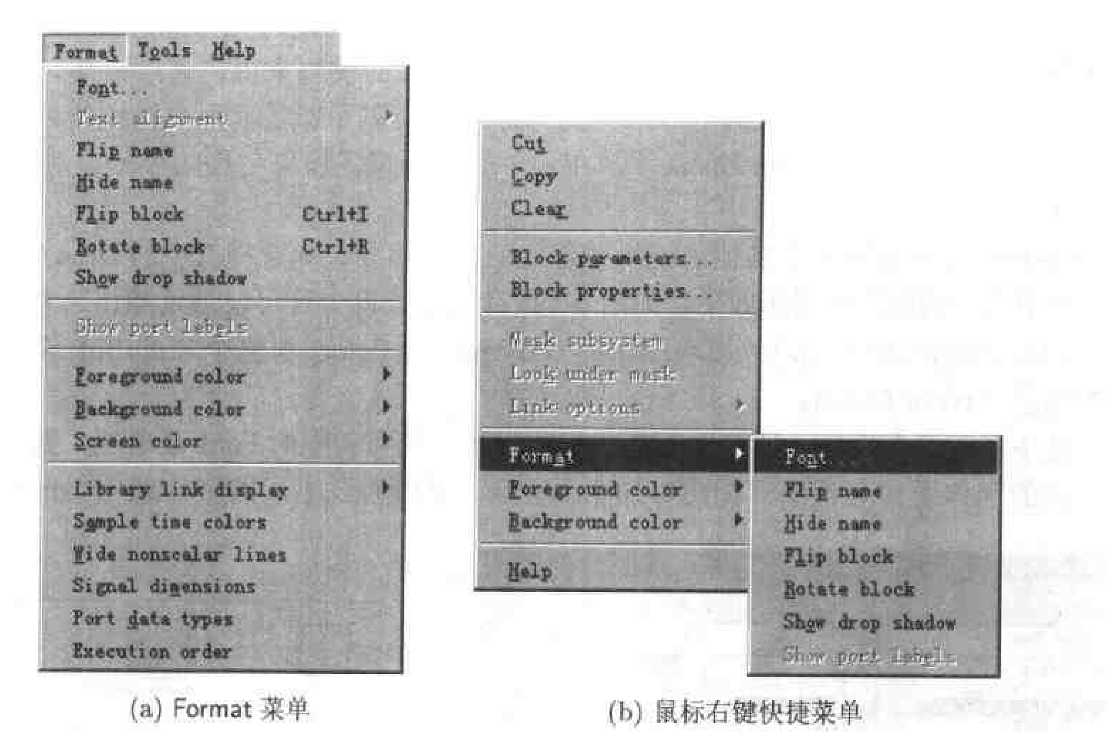
(a) 单个选中模块 (b) 一组选中模块

图 4-15 模块的选择

后拖动鼠标到区域右上角处释放，则整个区域内所有的模块将均被选中，如图 4-15 (b) 所示。另外，按下 Ctrl 键，再单击想选中的模块，则可以随意地同时选择多个模块。

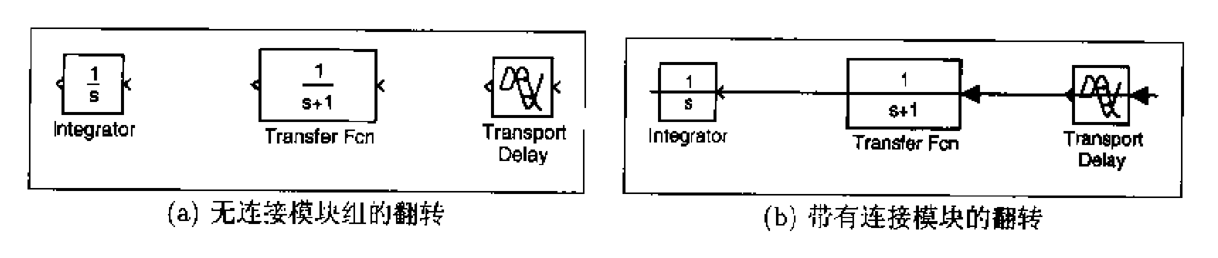
选中了一组模块，可以对之进行各种处理。例如若在反馈系统模型中，需要将处于反馈路径上模块的输入端和输出端掉换一下方向，则可以打开 Simulink 的 Format 菜单，如图 4-16 (a) 所示，选择其中的翻转子菜单 (Flip Block)，同样的任务还可以对选中的模块右击鼠标键，打开快捷菜单，再打开其中的 Format 子菜单，如图 4-16 (b) 所示，从中选择其中的 Flip Block 来完成，翻转后的模块组如图 4-17 (a) 所示。

如果选择的模块已经和其他的模块进行了连接，则将出现如图 4-17 (b) 所示的旋转效果，旋转后模块实际的连线仍是正确的，但布线显得有问题，需要手动地重新布线，



(a) Format 菜单　　(b) 鼠标右键快捷菜单

图 4-16　Simulink 下的模块格式设置菜单

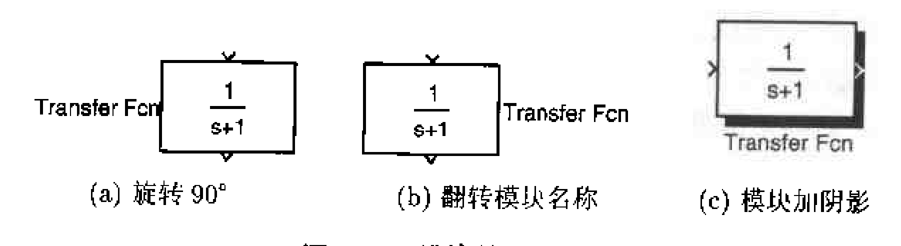


(a) 无连接模块组的翻转　　(b) 带有连接模块的翻转

图 4-17　模块的翻转

实施起来可能更麻烦，所以应该在连线之前进行模块旋转，再将旋转、翻转后的模块连接起来。

除了对模块进行旋转之外，有时为布线美观的需要，还需要将模块旋转 90°，这可以通过 Format 菜单中的 Rotate 菜单项来实现，单个模块的旋转效果如图 4-18 (a) 所示。



(a) 旋转 90°　　(b) 翻转模块名称　　(c) 模块加阴影

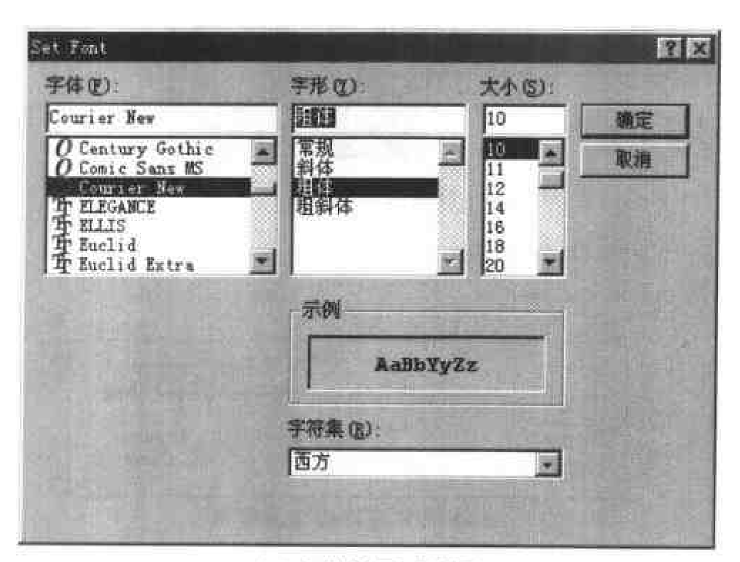
图 4-18　模块简单处理

还可以连续使用旋转命令，这样也能轻易地实现 270° 的旋转了。在 90° 的旋转中，将模

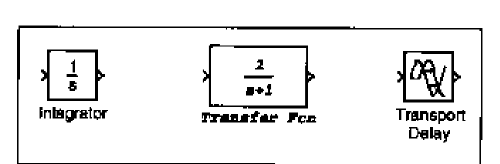
块的名称旋转到了模块的左侧，如果想将其移到模块的右侧，则可以选择 Flip Name 子菜单项，这将得出如图 4-18 (b) 所示的效果。对翻转的模块来说，通过模块名翻转的命令，将把模块的名称从模块的下方翻转到上方。如果您不想显示哪个模块的名称，则可以选中该模块，再选择 Hide Name 子菜单项。若想再恢复模块名称的显示，则选择 Show Name。

Format 菜单还提供了其他的模块或系统的修饰功能，如选择其中的 Drop Shadow 菜单项将给选中的模块加阴影效果，如图 4-18 (c) 所示。此外还可以通过颜色选项修改模块背景的 (Background Color)、模块标示的 (Foreground Color) 和整个 Simulink 模型窗口的背景颜色 (Screen Color)。

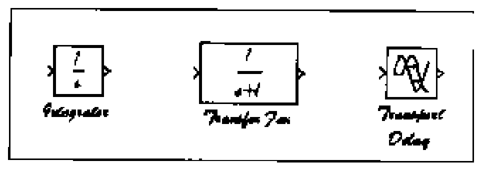
选中了若干个模块，还可以改变它们的字体，选择其中的 Font 菜单项，则将自动得出标准的字体设置对话框，如图 4-19 (a) 所示。可以通过不同的字体选项得出如图 4-19



(a) 字体设置对话框



(b) 选择 New Courier 字体



(c) 选择 Brush Script MT 字体

图 4-19　模块的翻转

(b) 和 (c) 所示的字体显示效果。注意，字体变化将同时体现在模块内部的字符表示与模块名称的表示。

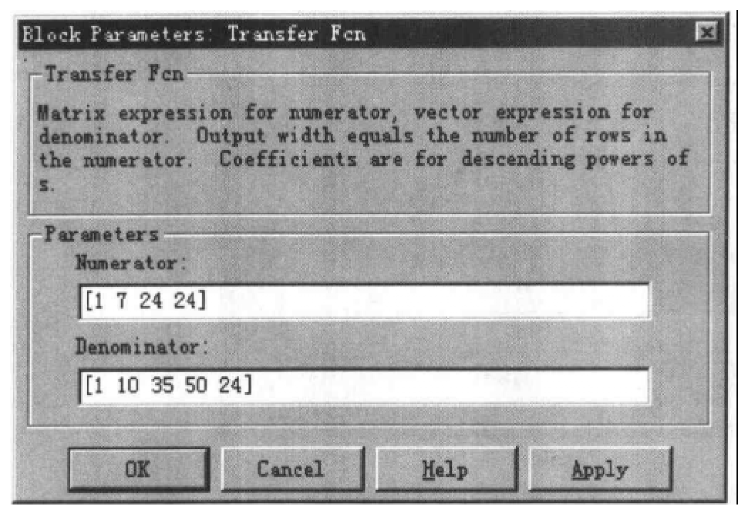
4.2.3　模块的参数修正

Simulink 在绘制模块时，只能给出带有默认参数的模块模型，这经常和想要输入的不同，所以要能够修改该模块的参数。例如前面例子中的传递函数模块的默认模型为 $G = 1/(s+1)$，而经常需要的模型参数显然和它的是不同的。例如想输入的模型为：

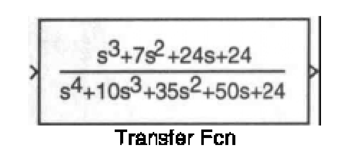
$$G(s) = \frac{s^3 + 7s^2 + 24s + 24}{s^4 + 10s^3 + 35s^2 + 50s + 24}$$

则可以双击该传递函数模块，在得出的如图 4-20 (a) 所示的对话框中，分别在分子输入编辑框和分母输入编辑框中输入系统的分子和分母参数，则可以最终获得修改后的系统模

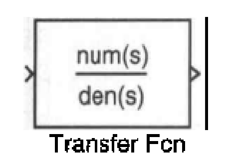
型。从给定的传递函数模型可以看出，系统传递函数的定义是一个分子多项式和一个分母多项式的比值，所以分别输入分子和分母多项式即可。在 MATLAB 和 Simulink 下，多项式可以由其系数按照降幂排列构成的向量来表示，这样 $s^3+7s^2+24s+24$ 可以表示成向量 [1,7,24,24]，这时在对话框的分子和分母编辑框中分别输入图 4-20 (a) 中的值，则可以正确表示该模块，单击 OK 按钮就可以将参数赋给该模块，这时模块的显示如图 4-20 (b) 所示。



(a) 传递函数参数对话框



(b) 确定参数显示



(c) 变量参数显示

图 4-20 模块简单处理

还可以用变量的形式表示这个模块，例如在对话框的两个编辑框中分别键入 num 和 den，则将会自动把模块的参数和 MATLAB 工作空间中的 num 和 den 两个变量建立起联系，这时模块的显示如图 4-20 (c) 所示。应该注意，在运行仿真之前，一定要在 MATLAB 的工作空间中给这两个变量赋值，否则将不能进行仿真分析。

积分器模块的模型更富变化，双击积分器模块，则将得出如图 4-21 所示的对话框，通过适当的设置，多种积分器的变化形式都可以表现出来。首先考虑给积分器设置初值，这可以由其中 Initial Value 编辑框的填写而完成，可以在该编辑框中填写常数或变量名，还可以为积分器的初值单独设置一个输入端口，接收其他信号的外部输入，这可以通过选择 Initial condition source 列表框中 external 选项，这时再按下 OK 按钮来实现，这样将可以得出如图 4-22 (a) 所示的模块表示，其中积分器的初值可以由外部信号设置。

积分器经常还可以设置复位信号，如果选择了 external reset 列表框中的 rising 选项，则将以外部信号的上升沿为准将积分器复位，其模块表现形式如图 4-22 (b) 所示，这样就可以给积分器加一个复位信号了。在该列表框还可以选择下降沿等作为复位控制信号。

在新版本的 Simulink 中，积分器本身还可以带有输出饱和，选择了 Limit output 复选框，并设置了饱和的上下限，则可以在模块的输出后加饱和处理，其模块表示形式如图 4-22 (c) 所示。通过选择各种附加选项，则可以得出如图 4-22 (d) 所示的模块效果，所以可以在仿真中充分利用积分器的各种选项，解决各种特殊的问题。

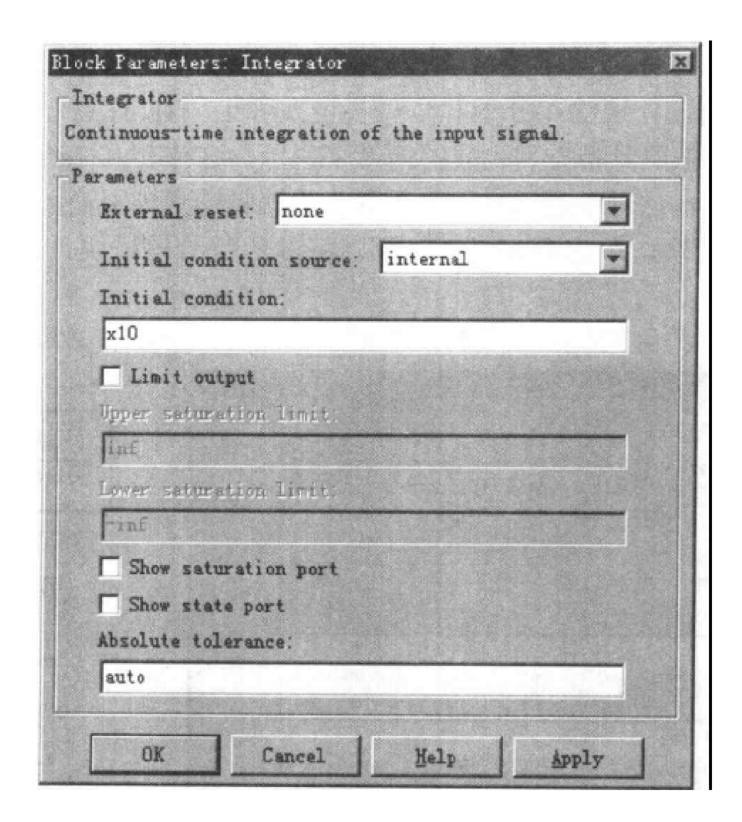
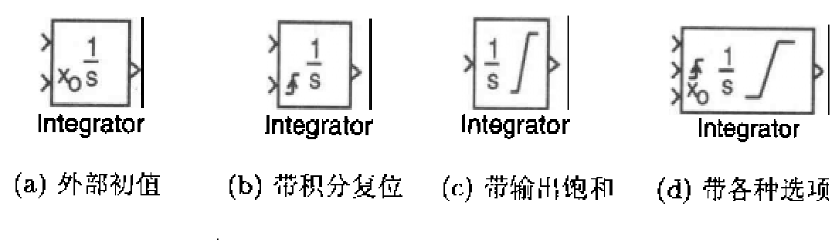


图 4-21 积分器模块的参数修改



(a) 外部初值 (b) 带积分复位 (c) 带输出饱和 (d) 带各种选项

图 4-22 积分器的其他表现形式

4.2.4 Simulink 模块的联机帮助系统

和 MATLAB 其他内容一样，Simulink 也提供了较完善的联机帮助系统，选中一个模块，选择 Help | Help on the selected block 菜单项或右击该模块，并在快捷菜单中选择 Help，则将打开一个如图 4-23 所示的帮助窗口。如果 MATLAB 的文档光盘未放入光驱，则将给出错误信息，无法进行联机帮助。

打开了所关心模块的帮助页面，还可以通过该页面直接访问相关的页面。这样的帮助系统使用起来还是较方便的。

4.2.5 Simulink 模型的输出与打印

在 Simulink 的模型编辑窗口下，选择 File | Print，则将给出如图 4-24 所示的对话框，按下 OK 按钮，则将自动将整个 Simulink 模型按照默认的格式在打印机上打印出

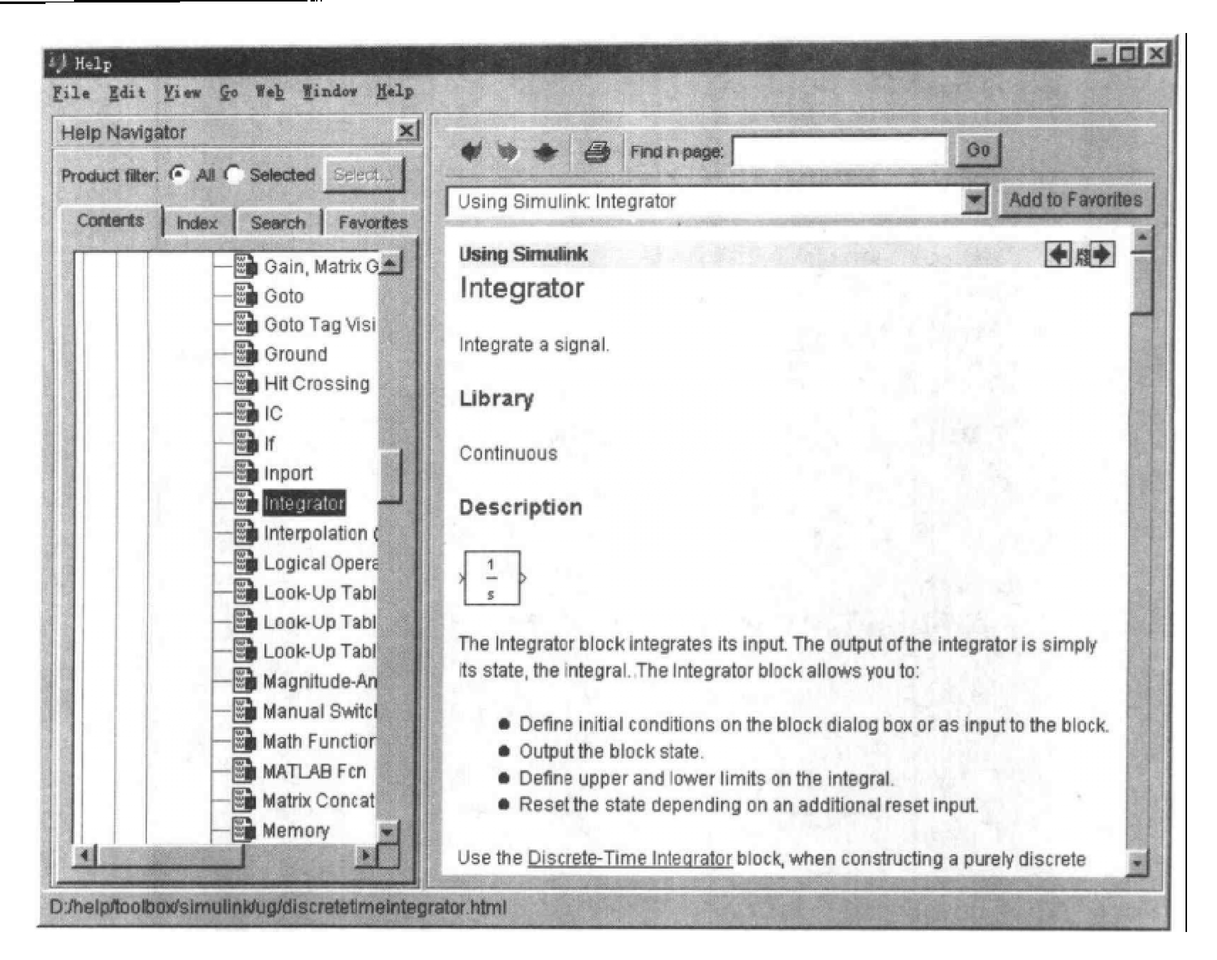


图 4-23 Simulink 模块的联机帮助系统

来。该对话框有各种各样的选项，如选择打印当前模型、当前模块及上级模块、下级模块等。另外还可以通过 Properties (属性) 按钮选择打印的其他属性，比如“打印方向” (Orientation) 中的“横向” (Landscape) 和“纵向” (Portrait) 等。当然，由于其属性对话框的标签太多，不宜寻找属性，所以这些参数的设置更适合通过 File | Page Setup 菜单对应的对话框来设置。

还可以通过 **print** 命令来打印或将存储文件，其格式为：

```
print -s -d类型 文件名
```

其中“类型”可以选择各种各样不同的文件类型，用 **help print** 列出，其中 **print -s -deps myfile** 命令将按封装的 PostScript 格式将图形存成 myfile.eps 文件。如果不使用 **-s** 选项，则可以将 MATLAB 图形窗口中的图形存成 eps 文件。

Simulink 模型可以由该窗口的 Edit | Copy model to clipboard 菜单将整个模型复制到 Windows 的剪贴板中，以便其他软件能直接调用。

4.2.6 启动仿真环境

建立好了 Simulink 模型后就可以启动仿真过程了。最简单的方法当然是按下

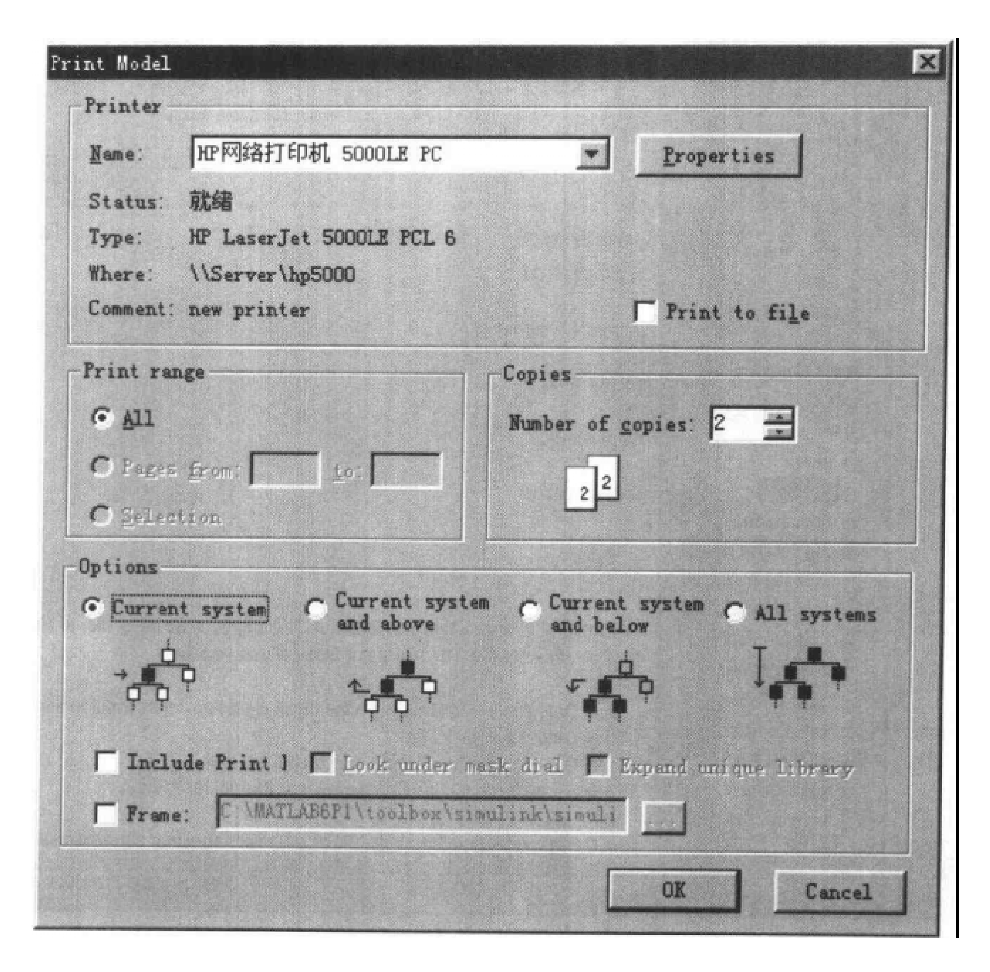


图 4-24 模型打印对话框

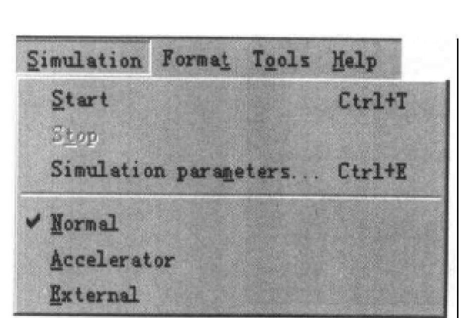
Simulink 工具栏下的“启动仿真”按钮了。启动仿真过程后将以默认参数为基础进行仿真，而用户还可以自己设置出需要的控制参数。仿真控制参数可以由 Simulation | Simulation parameters 菜单项来选择，如图 4-25 (a) 所示。选择了该菜单项后，将得到如图 4-25 (b) 所示的对话框，用户可以从中填写相应的数据，控制仿真过程。

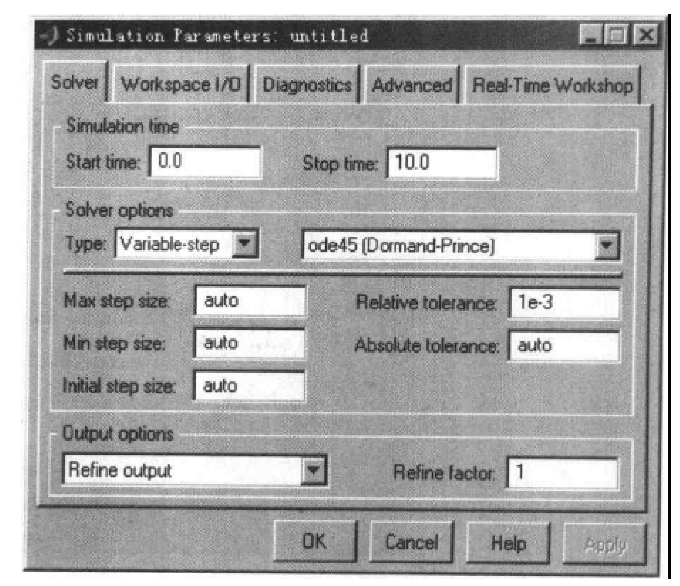
在图 4-25 (b) 的对话框中有 5 个标签[①]，默认的标签为微分方程求解程序 Solver 的设置，在该标签下的对话框主要接受微分方程求解的算法及仿真控制参数。

- **仿真算法选择** 通过该对话框可以由 Solver options 栏目选择不同的求解算法，定步长下支持的算法如图 4-26 (a) 所示，而变步长下的算法如图 4-26 (b) 所示。一般情况下，连续系统仿真应该选择 ode45 变步长算法，对刚性问题可以选择变步长的 ode15s 算法，离散系统一般默认地选择定步长的 discrete (no continuous states) 算法，而在仿真模型中含有连续环节时注意不能采用该仿真算法，而可以采用诸如四阶 Runge-Kutta 法这样的算法来求解问题。

  定步长算法的步长应该由 Fixed step size 编辑框中填入参数指定，一般还可以选择 auto，依赖计算机自动选择步长，而变步长下建议步长范围使用 auto 选项。在实

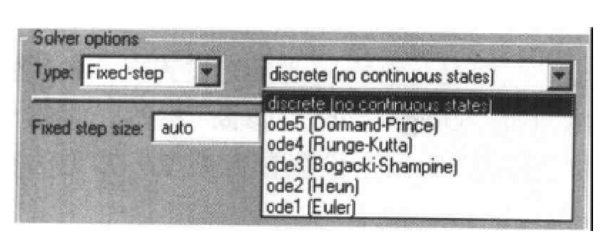
[①] 在未安装实时工具箱时，将不出现 Real-Time Workshop 标签，该对话框只有前 4 个标签。

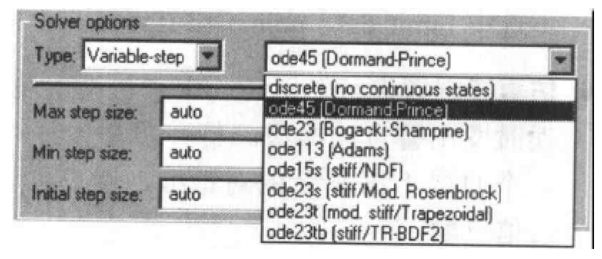




(a) Simulation 菜单　　(b) 仿真参数设置对话框

图 4-25　Simulink 仿真菜单和对话框





(a) 定步长仿真算法　　(b) 变步长仿真算法

图 4-26　Simulink 仿真算法选择

时工具中要求必须选用定步长的算法。

- **仿真区间的设置**　在该对话框中还可以修改仿真的初始时间和终止时间；另外，用户还可以利用 Sinks 模块组中的 Stop 模块来强行停止仿真过程。
- **输出信号的精确处理**　由于在仿真中经常采用变步长算法来完成，故有时会发现输出信号过于粗糙，所以要对得出的输出进行更精确的处理，这就需要在 Output options 栏目中选择 Refine output 选项，并将其 Refine factor 选项选择一个大于 1 的数值。
- **MATLAB 工作空间设置**　单击对话框中的 Workspace I/O 标签，则打开如图 4-27 所示的对话框，可以看出，在默认状态下，时间和输出信号都将写入 MATLAB 的工作空间，分别存入 tout 变量和 yout 变量。在实际仿真中建议保留这两个选项。如果想获得系统的状态，则还可以选中 xout。

在该对话框中，还可以选择输出向量的最大长度，默认值为 1000，即保留 1000 组数据，如果因为步长过小，则实际计算出来的数据量很大，超过选择的值，这样在

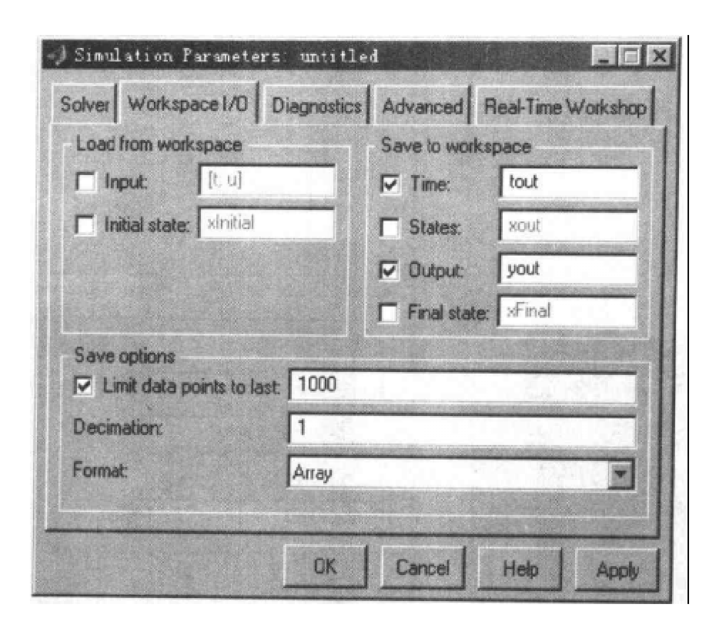


图 4-27　工作空间输入输出对话框

MATLAB 工作空间中将只保存 1000 组最新的数据。如果想消除这样的约束，则可以不选中 Limit data points to last 复选框即可。

- **仿真错误警告**　在 Simulink 中可能出现一些错误情况，这就需要事先设置出现各类错误时发出警告的等级。打开仿真参数对话框中的 “Diagnostics” (诊断) 标签，将得出一个如图 4-28 所示的对话框，用户可以对可能发生的错误设置警告类型。常见的警告信息如：

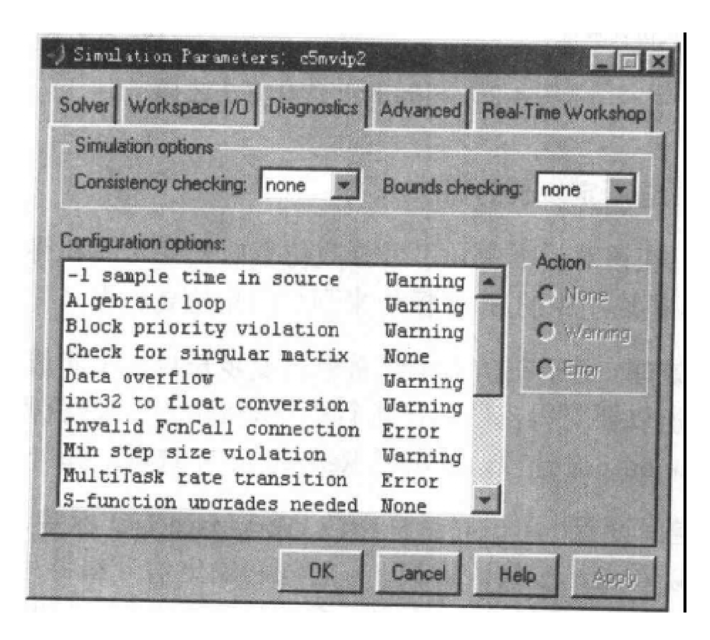


图 4-28　仿真错误诊断对话框

◇ -1 sample time in source (在输入模块中使用 -1 采样周期)　在离散系统建模中，通

常可以将一些模块采样周期设置为 −1，表示可以继承其输入信号的采样周期。但如果输入模块的采样周期设置为 −1，则会发生错误，使得整个系统中的模块采样周期无法确定。

◇ Algebraic loop (代数环) 经常由于某个或某些模块的输出信号作为输入信号，再直接传递到该模块的输入端而生成一种环路。如果产生代数环的是由线性模块构成，则可以手动地化简相应的模块就能消除代数环，后面将通过例子来演示。

◇ Unconnected block output (输出端口未连接) 在模型中如果有的模块输出端口若未连接到其他模块，则将给出错误信息。解决这样问题的方法是将悬空的端口连接到输出池的终结模块 (Terminator) 上。类似的，还将检测 Unconnected block input (输入端口未连接)、Unconnected lines (未连接的线) 等，前者可以给其输入一个零信号，如 Ground 模块，后者可以删除不必要的连线。其实存在悬空端子并不会影响仿真结果，所以可以取消对悬空端子的检测。

◇ Min step violation (小于最小步长) 在变步长计算中如果自动选择的步长小于预先指定的最小步长，则将发生这样的错误，这时需要研究原来的最小步长选择是否合理，另外，应该看看约定的计算精度是否过高。

◇ int32 to float conversion (32 位整数转换浮点数错误) 等数据类型转换错误，类似的还有 Unneed conversion (不必要的类型转换)、Vector/matrix conversion (向量、矩阵类型转换) 等。

- **高级仿真属性设置** 单击对话框中的 Advanced 标签，则将得出如图 4-29 所示的对话框，在该对话框中用户可以选择一些进一步的属性，更好地控制仿真过程。在默认情况下：

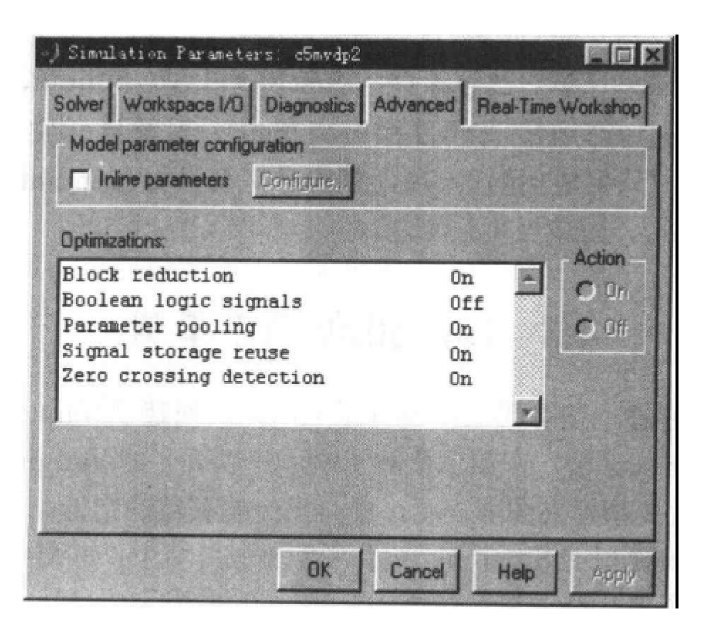


图 4-29 高级属性设置对话框

◇ Block reduction 选项可以作模块的简化，这样可以加速仿真过程。

◇ Boolean logic signals 选项将允许在仿真中使用逻辑信号。

◇ Zero-crossing detection 选项将在仿真过程中精确计算过零点。

- **实时工具对话框** 单击仿真参数对话框中的 Real-Time Workshop 标签，则可以得出如图 4-30 所示的对话框，在该对话框中可以设置若干实时工具中的参数，如果没有安装实时工具，则将不出现此对话框标签。

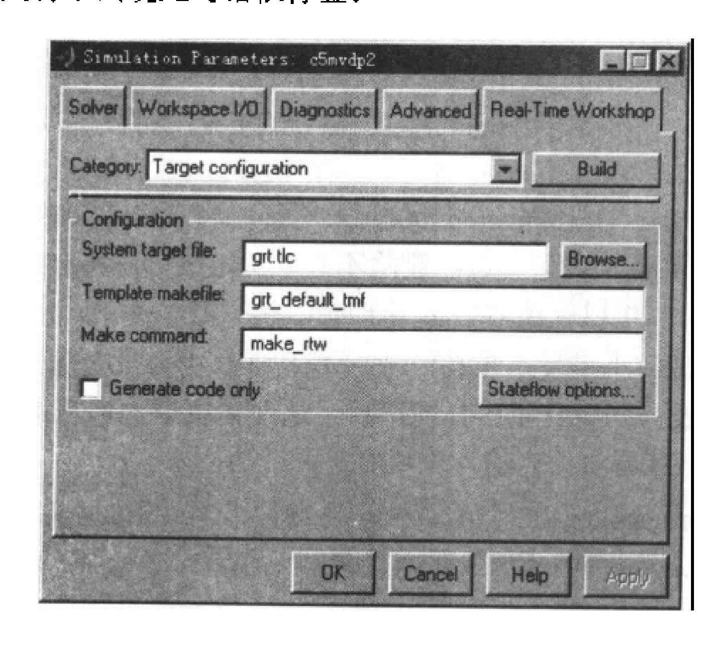


图 4-30　实时控制设定对话框

在 Simulink 的实时工具中引入了目标语言编译 (target language compiler，简称 TLC) 的概念，允许用户自己指定目标语言，最终将 Simulink 模型翻译成优化的代码，经过编译过程最终生成可执行文件，脱离 MATLAB/Simulink 环境执行。

在该对话框中允许用户选择目标语言模板、系统目标文件等，如果选择了 Generate code only (只生成代码) 选项，则实时工具只将 Simulink 模型翻译成目标语言代码，不进行编译、生成可执行文件。

4.3　Simulink 模型举例

在这一节中，将通过一些有代表性的例子来演示如何建立 Simulink 模型，并介绍如何对给定的模块进行仿真分析。在第 1 个例子中，将以著名的 Van der Pol 方程为例演示如何将给出的微分方程模型建立图形表示，并得出一些有益的结论；第 2 个例子中将介绍一个线性框图形式给出模型的 Simulink 表示方法及仿真结果，并对不同控制器参数进行仿真分析。第 3 个例子将介绍非线性模型的 Simulink 描述与仿真，并演示非线性参数和输入幅值变化时仿真的结果。第 4 个例子将介绍采样系统的 Simulink 表示及仿真，演示在不同采样周期下系统的性能。在第 5 个例子中将演示代数环的结构及避免的方法，第 6 个例子将演示数字逻辑电路在 Simulink 下的搭建和仿真方法。

〖例 4.1〗 首先考虑例 3.30 中描述的 Van der Pol 方程

$$\dot{x}_1 = x_2,\ \ \dot{x}_2 = -\mu(x_1^2 - 1)x_2 - x_1$$

第 1 个方程可以认为是将 $x_2(t)$ 信号作为一个积分器的输入端，这样积分器的输出则将成为 $x_1(t)$ 信号、类似地，$x_2(t)$ 信号本身也可以认为是一个积分器的输出，在积分器的输入端信号应该为 $-\mu(x_1^2 - 1)x_2 - x_1$，在构造该信号时还应该使用信号乘积的处理，这样可以按图 4-31 所示的格式建立起描述该微分方程的模型。除了上述的各个模块后，别忘了还需要添加结果输出模块，这里使用输出端口模块输出结果。

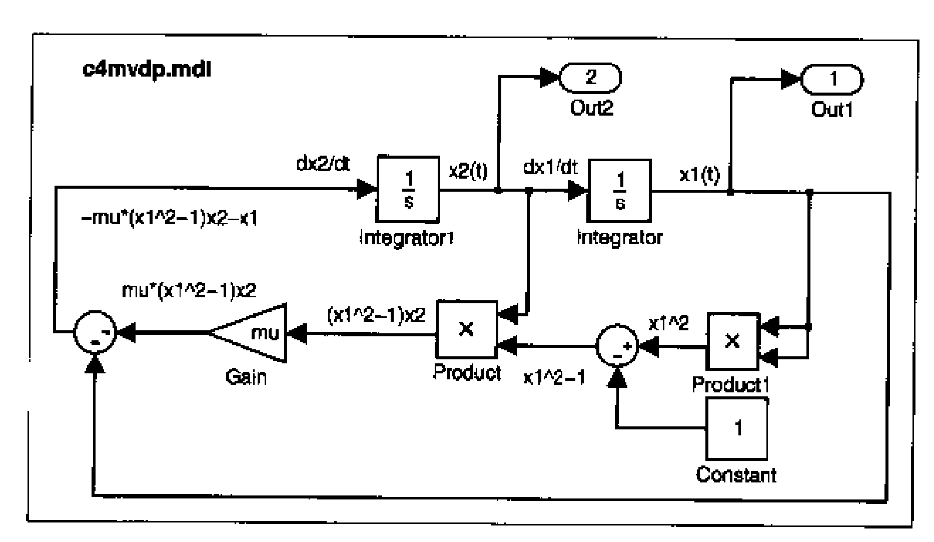


图 4-31 Van der Pol 方程的 Simulink 表示

可以看出，在系统框图中，除了各个模块及其连接之外，还给出了各个信号的文字描述。在 Simulink 模型中加文字描述的方式也很简单，在想加文字说明的位置双击鼠标，则将出现字符插入的标示，这时可以将任意的字符串写到该位置即可。文字描述写到模型中后，则可以用鼠标单击并拖动到指定位置。

从这个例子可见，很多微分方程实际上应该是可以由 Simulink 用图示的方法完成的，可以将这样的思想应用于更复杂系统的建模。在本模型中还涉及到几个参数，需要在 MATLAB 中予以赋值，如 mu 的值与两个积分器的初始值 x01 和 x02。双击加法器模块，在得到的对话框中填写 |-- 则可以得出两个减号的加法器，其中第一个 | 号表示上面的入口为空，后面两个减号表示减号输入端。

输入了适当的参数后，还需要在 MATLAB 命令窗口中给出如下命令

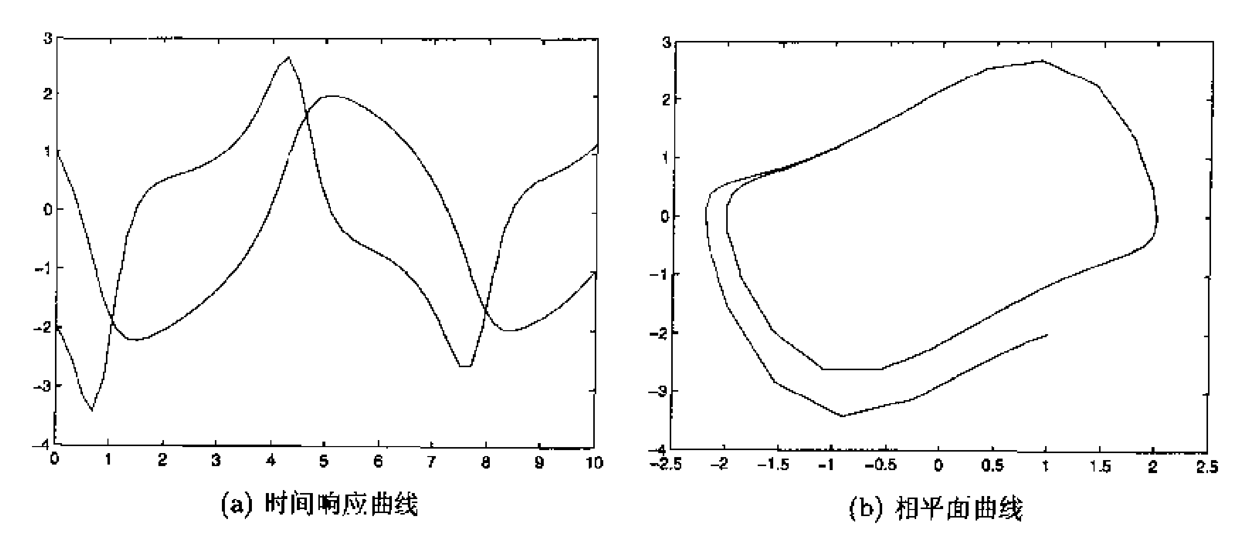
```
>> mu=1; x01=1; x02=-2;
```

这时就可以启动仿真命令了，如按下 Simulink 工具栏中的启动按钮，或选中 Simulation/Start 菜单项，经过暂短的仿真过程，则仿真结果将赋给 MATLAB 工作空间内的保留变量 tout 和 yout。在 MATLAB 命令窗口中给出绘图命令，

```
>> plot(tout,yout), figure, plot(yout(:,1),yout(:,2))
```

则将分别得出如图 4-32 (a) 和 (b) 所示的时间响应曲线和相平面曲线。

还可以修改一下系统的输出方式，将 x_1 和 x_2 信号分别接入 x-y 示波器的两个输入端，如图



(a) 时间响应曲线 (b) 相平面曲线

图 4-32 Van der Pol 方程的仿真结果

4-33 所示，再双击示波器图标，则将得出如图 4-34 (a) 所示的对话框，在该对话框下可以填写示

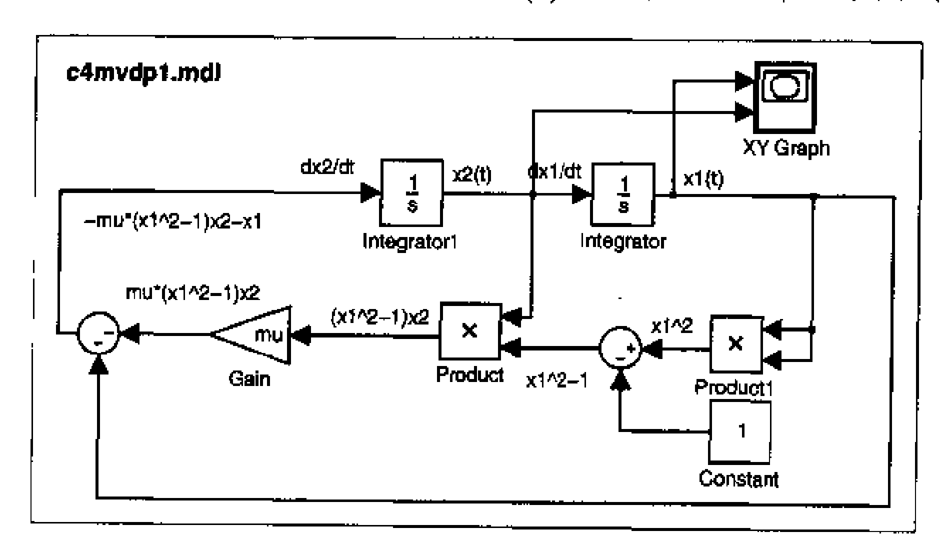


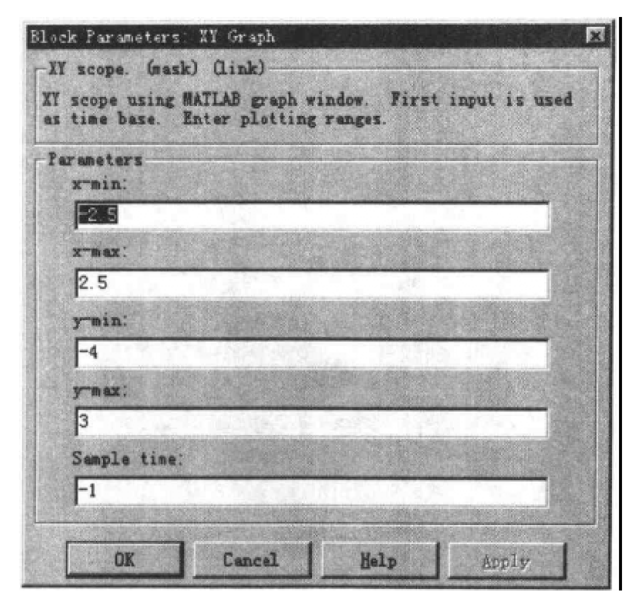
图 4-33 带 x-y 示波器的 Simulink 模型

波器的 x-y 轴范围，可以按照图中的方式填写这些参数，关闭对话框后启动仿真过程，则将得出如图 4-34 (b) 所示的相平面图，可以看出，这里得到的图形和例 3.30 中的完全一致。

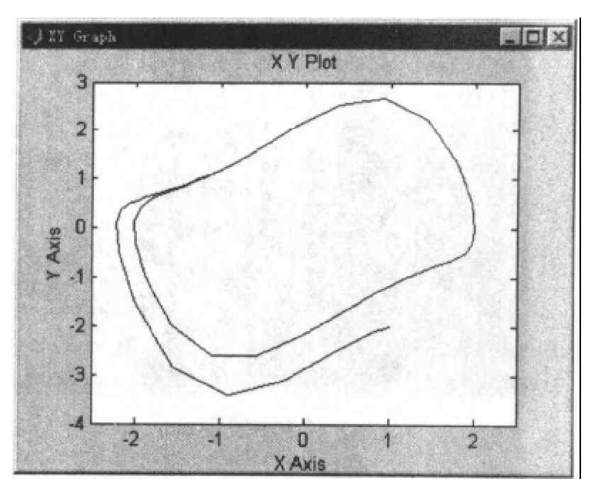
还可以使用另一种示波器——常规示波器来显示得出的结果。因为示波器默认图标只接收一路信号，而我们想同时绘制两条状态曲线，这样应该使用 Signals & System 组中的 Mux 模块将两路输入进行向量化，混合成一路输出，直接连接到示波器上，如图 4-35 所示。

再进行仿真，就可以得出如图 4-36 (a) 所示的仿真结果。该结果中，坐标轴是按默认的格式选择的，对本例来说自动选择的 y 轴不甚理想，可以通过示波器自己的工具栏来对曲线进行放大处理，如单击工具栏中的“望远镜”按钮，最终得出如图 4-36 (b) 所示的形式。

很多微分方程都是可以这样地表示成图形形式的，但比较起来，用第 3 章中的 MATLAB 语句描述的微分方程求解应该更简单、直观，且不易出错、所以对一般微分方程来说，最好采用语



(a) 示波器参数设置



(b) 示波器显示结果

图 4-34 x-y 示波器的相平面显示

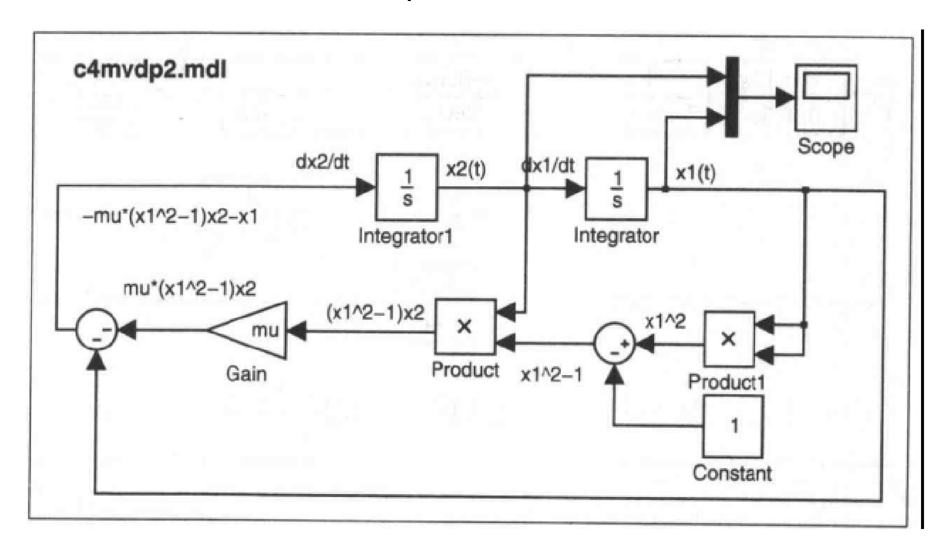
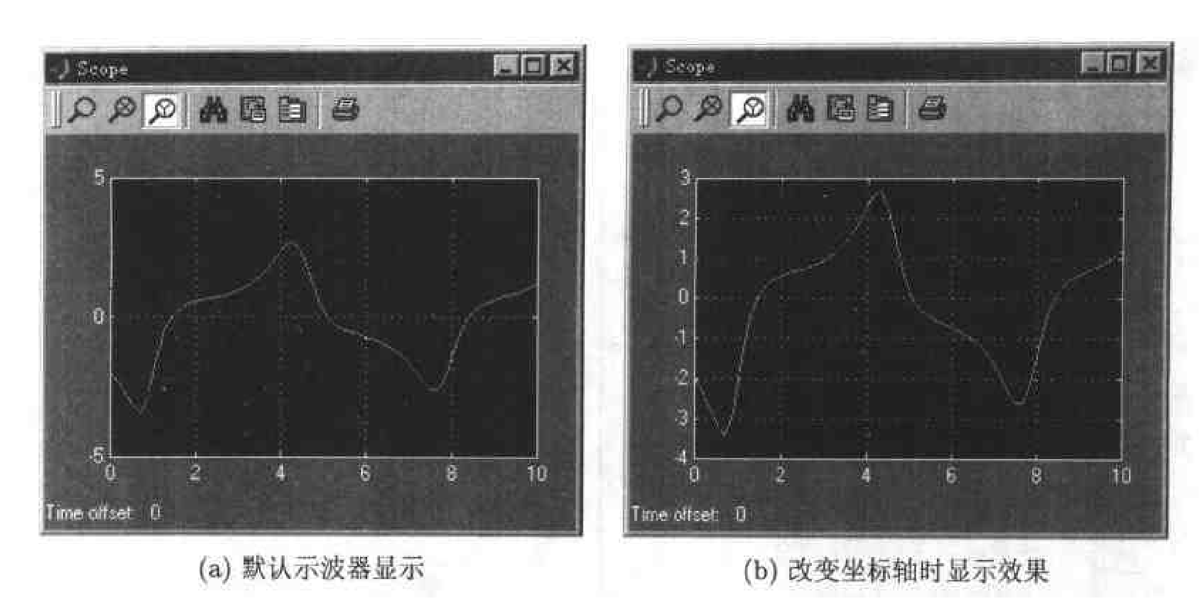


图 4-35 带示波器的 Simulink 模型

句的方式求解微分方程。

〖例 4.2〗 考虑直流电机拖动系统[42]，如图 4-37 所示。可以按照图 4-38 中给出的方式构造系统的 Simulink 模型，在该系统中，输入端采用两个信号的叠加形式，其中一个是实际输入的阶跃信号，另一个是系统的输入端子。在 Simulink 中，默认的阶跃输入模块的跳跃时间为 1，而在控制系统研究中习惯将其定义为 0，故可以修改其参数。要修改模型中其他模块的参数，则可以直接双击其图标，然后在得出的对话框中填入适当的数据即可。

得出了系统的 Simulink 模型后，则可以对该系统进行仿真研究。如选择其中的 Simulation | Start 菜单项，则可以立即得出仿真结果，该结果将自动返回到 MATLAB 的工作空间中，其中时间变量名为 tout，输出信号的变量名为 yout。使用命令



(a) 默认示波器显示　　(b) 改变坐标轴时显示效果

图 4-36　示波器的时间响应显示

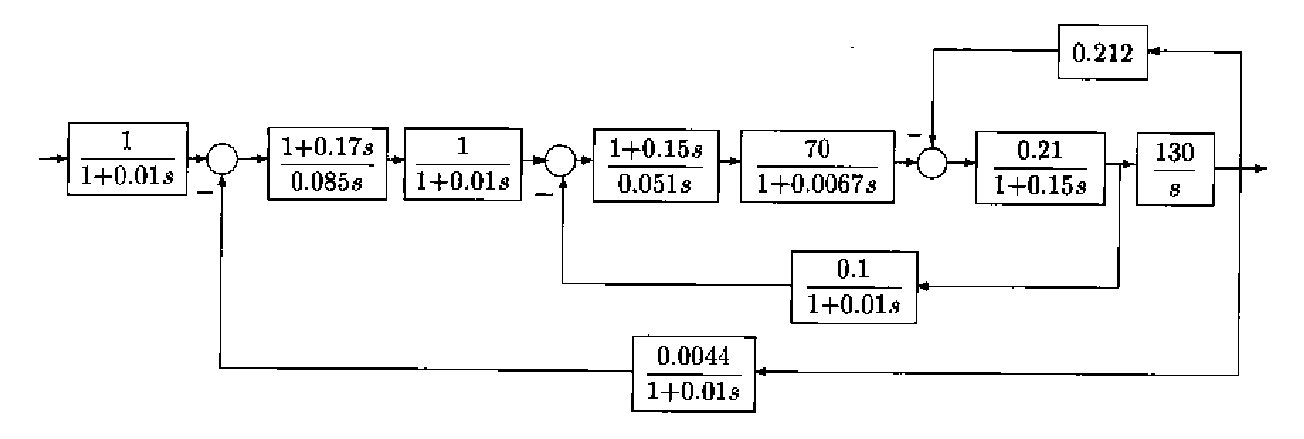


图 4-37　一个直流电机拖动系统的例子

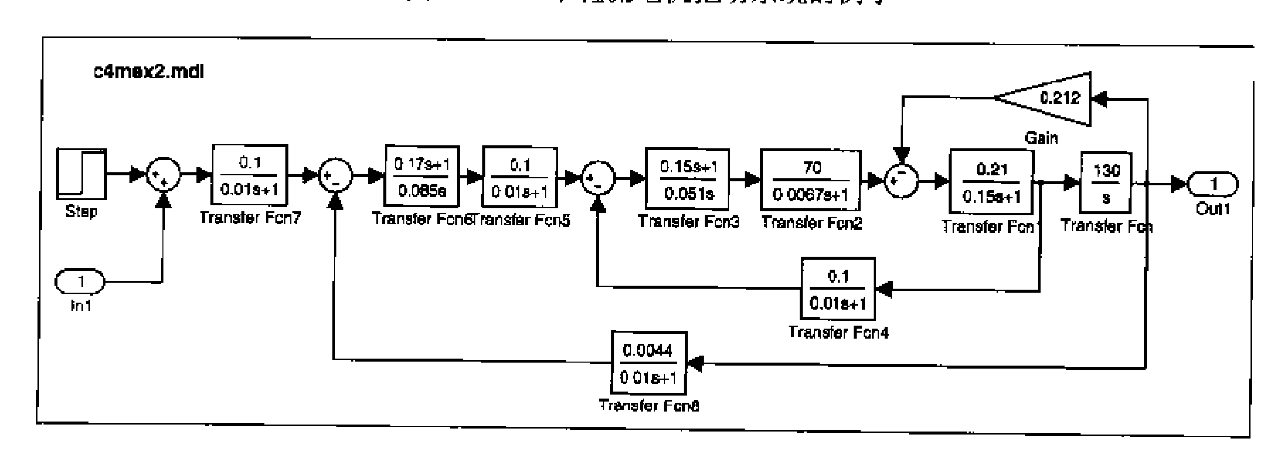
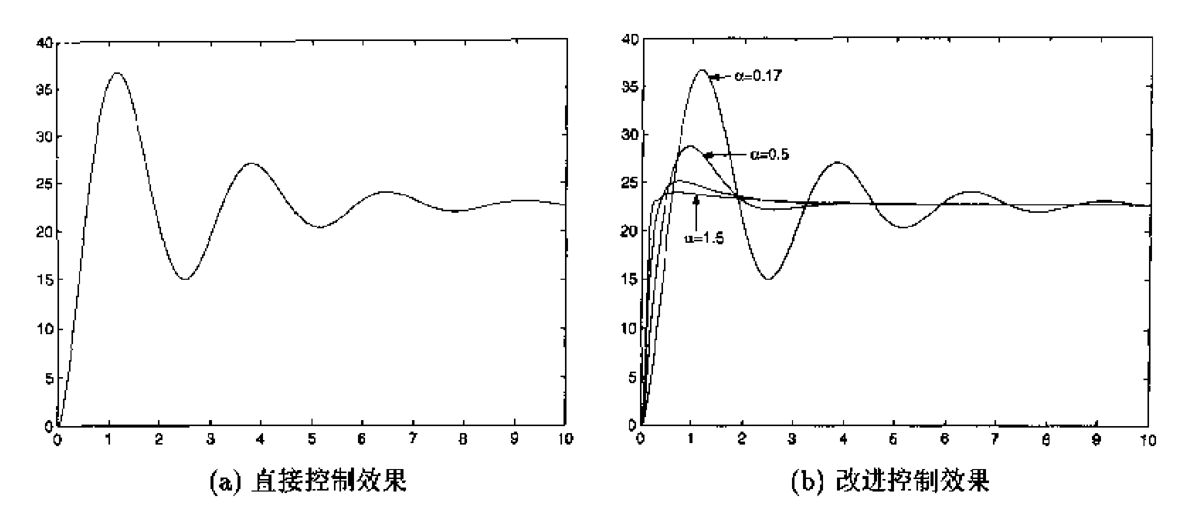


图 4-38　电机拖动模型 Simulink 表示

```
>> plot(tout,yout)
```

则可以立即绘制出系统的阶跃响应曲线，如图 4-39 (a) 所示。从响应曲线看，该曲线不是很理



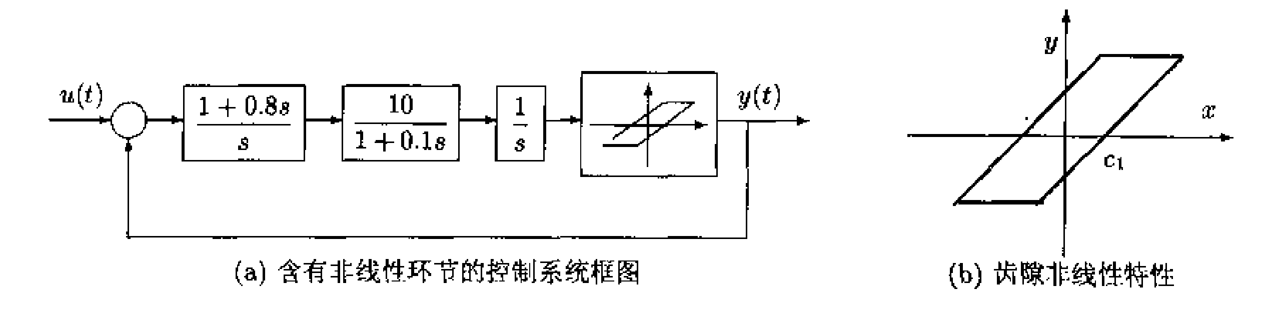
(a) 直接控制效果　　(b) 改进控制效果

图 4-39　输出信号的响应曲线

想，可以将外环的 PI 控制器参数调整为 $(\alpha s+1)/0.085s$，并分别选择 $\alpha=0.17,0.5,1,1.5$，则可以得出如图 4-39 (b) 所示的结果。可以看出，如果选择 PI 控制器为 $(1.5s+1)/0.085s$ 则能得到较满意的效果。

可以看出，在 Simulink 仿真环境中，可以方便地修改系统的参数。系统模块的参数既可以通过 MATLAB 下的命令修改，也可以在仿真过程中打开参数对话框的方式进行修改。通过命令修改的方式在第 6 章中还将详细介绍。

〖例 4.3〗 含有磁滞回环非线性环节的控制系统框图如图 4-40(a) 所示，其中磁滞回环非线性环节的特性如图 4-40(b) 所示。



(a) 含有非线性环节的控制系统框图　　(b) 齿隙非线性特性

图 4-40　非线性系统框图

可以看出，这里除了线性环节外，还有一个磁滞回环非线性环节，该环节可以用 Simulink 非线性模块库中的 Backlash 模块表示。这时可以容易地由 Simulink 建立起仿真模型，如图 4-41 所示。在仿真之前还应该给磁滞宽度 c_1 赋值：c1=1，并设置终止仿真时间为 3，这样可以启动仿真过程，仿真过程结束后，则将自动在 MATLAB 工作空间中生成两个变量 —— tout 和 yout，用下面的语句将能绘制出系统的阶跃响应曲线，如图 4-42 (a) 所示。

```
>> plot(tout,yout)
```

对不同的 c_1 取值，可以用同一个 Simulink 模型结构对之进行仿真，并将结果在 MATLAB 语言中

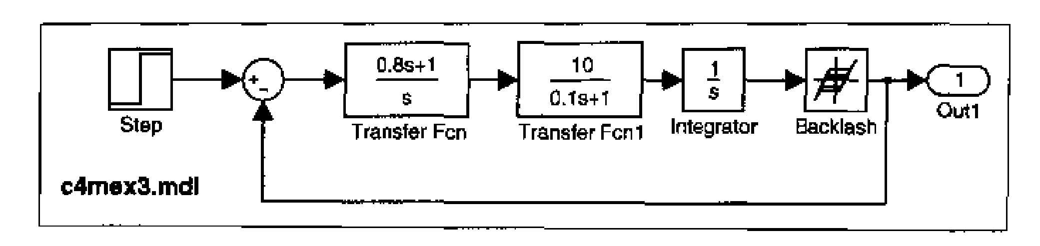
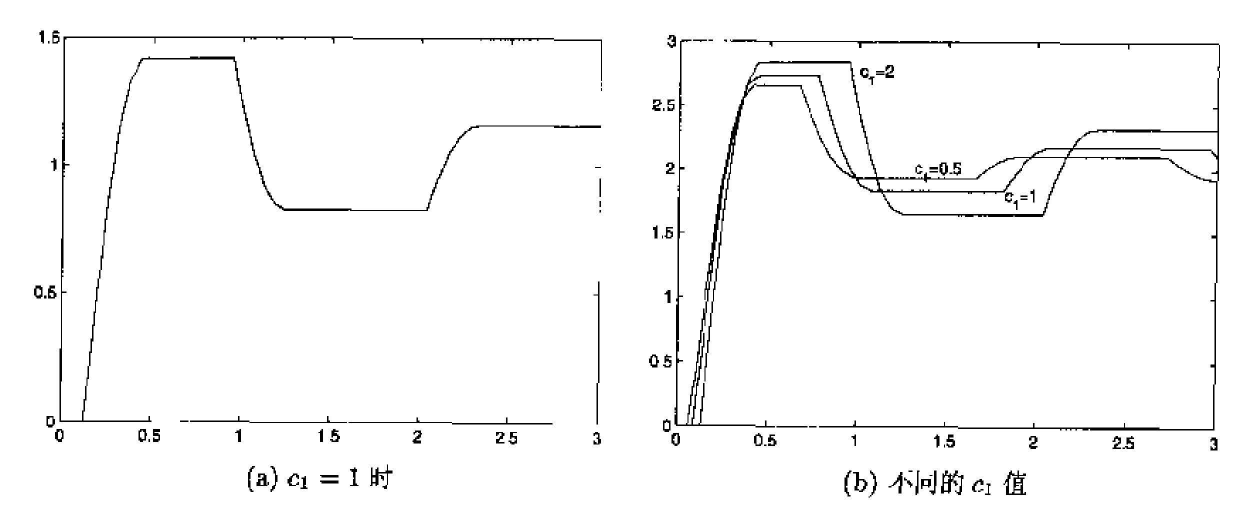


图 4-41　非线性系统的 Simulink 模型表示



(a) $c_1 = 1$ 时　　(b) 不同的 c_1 值

图 4-42　磁滞回环系统的阶跃响应

绘制出来，如图 4-42 (b) 所示。在第一次绘制图形后，应该给出 hold on 命令，使得所有的阶跃响应曲线都绘制在同一坐标轴下，以便比较。

和线性系统不同，如果输入的幅值增大或减小，原来系统响应曲线的形状将可能出现不同。例如：若输入信号的幅值变成 2，则对不同的 c_1 取值，可以由 Simulink 模型得出仿真结果，如图 4-43 所示。

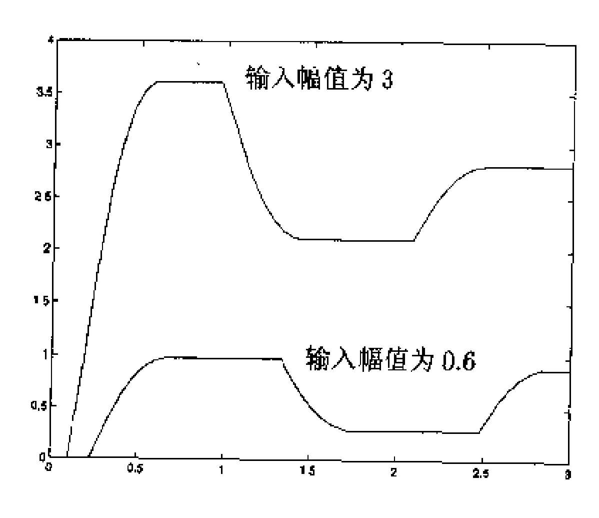


图 4-43　输入幅值改变后仿真结果

从这个例子可以看出，复杂的非线性环节在 Simulink 也可以容易地进行仿真，还可以充分利

用 Simulink 的功能改变非线性环节的参数，将结果在图形中显示出来。

〖例 4.4〗 已知采样系统的结构图如图 4-44 所示。可以建立起其 Simulink 模型，如图 4-45 所示。

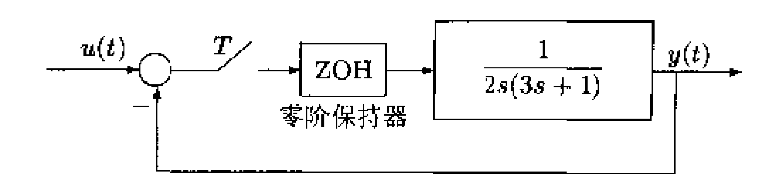


图 4-44 采样控制系统结构图

注意，这里传递函数是分为两个部分 g1 和 g2 建立的，作者建议在建立系统时尽量少采用人工的运算，减少出错的机会。在该系统中使用了 Discrete 模块库中的零阶保持器图标，并设置其采样

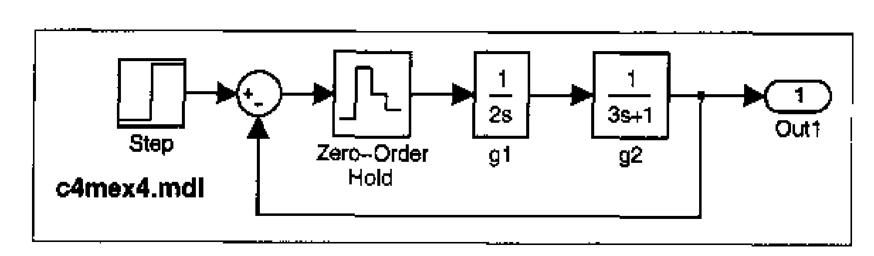


图 4-45 采样系统的 Simulink 表示

周期为 0.1。仿真完成后，可以由 MATLAB 命令 `stairs(tout,yout)` 可以得出输出信号阶跃响应曲线，如图 4-46 所示。

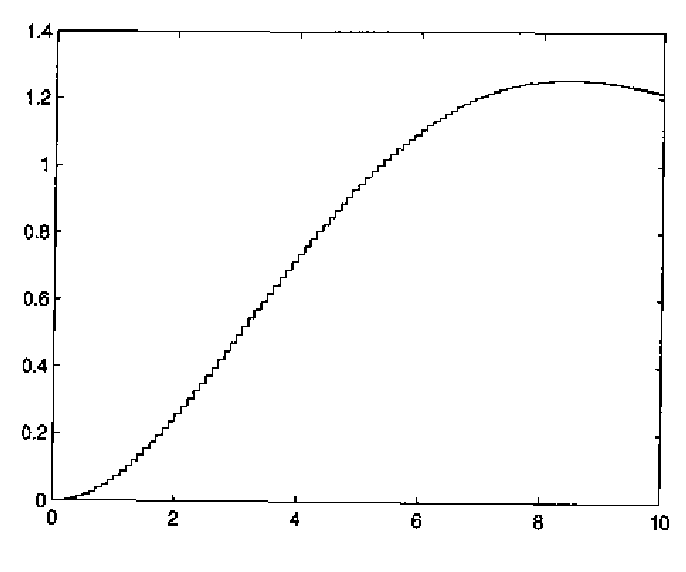
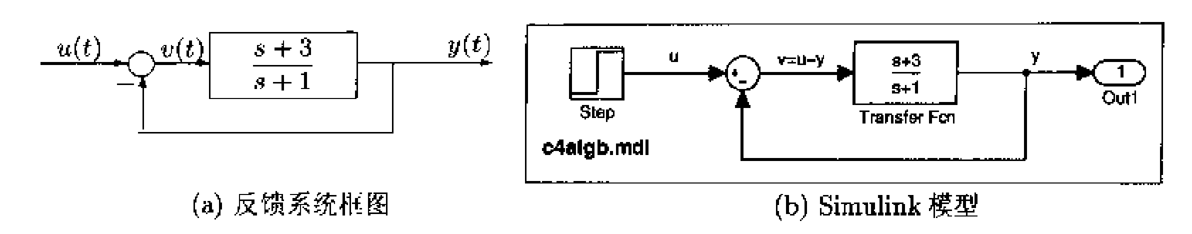


图 4-46 采样系统阶跃响应

〖例 4.5〗 考虑一个简单的线性系统模型，如图 4-47 (a) 所示。可以绘制出如图 4-47 (b) 所示的 Simulink 框图，对这样的系统进行仿真，将在 MATLAB 的命令窗口中给出警告信息：

```
Warning: Block diagram 'untitled' contains 1 algebraic loop(s).
```

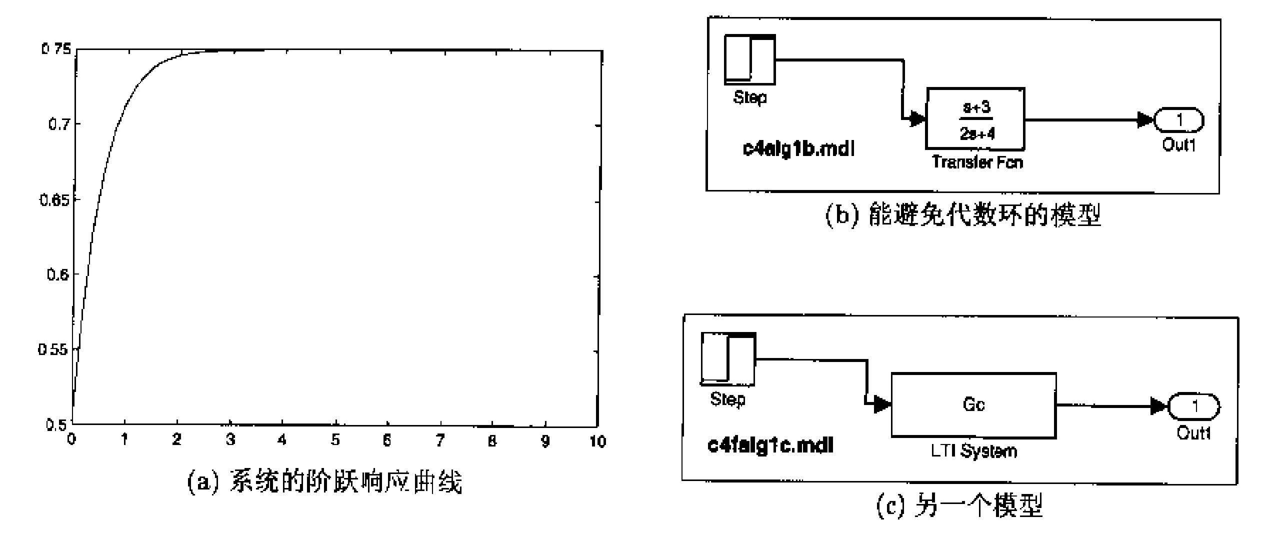


(a) 反馈系统框图　　(b) Simulink 模型

图 4-47　简单反馈系统模型

```
Found algebraic loop containing block(s):
    'untitled/Transfer Fcn'
    'untitled/Sum' (algebraic variable)
```

说明该模型在仿真时含有代数环 (algebraic loop)。分析图 4-47 (b) 中的 Simulink 框图，可以看出传递函数模块的输入为 $u-y$，而 y 又是其自己的输出，这样就构成了代数环。其实即使代数环存在，Simulink 仍然能得出系统的响应，如图 4-48 (a) 所示。虽然在计算中给出了警告信息，但得出的结果还是正确的，说明 Simulink 能正确地处理一般的代数环问题。



(a) 系统的阶跃响应曲线　　(b) 能避免代数环的模型　　(c) 另一个模型

图 4-48　简单反馈系统模型

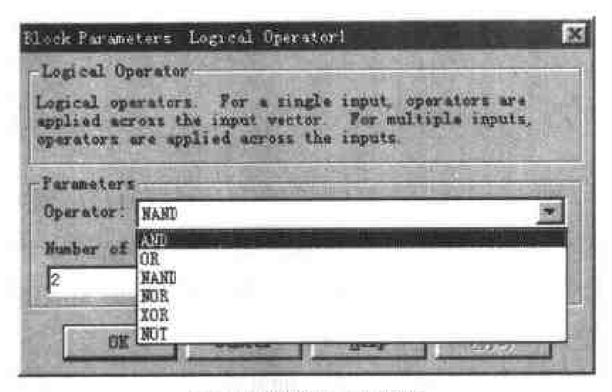
要想避免代数环，可以将整个反馈回路进行手工简化。

```
>> Go=tf([1 3],[1 1]); Gc=feedback(Go,1)
Transfer function:
 s + 3
-------
2 s + 4
```

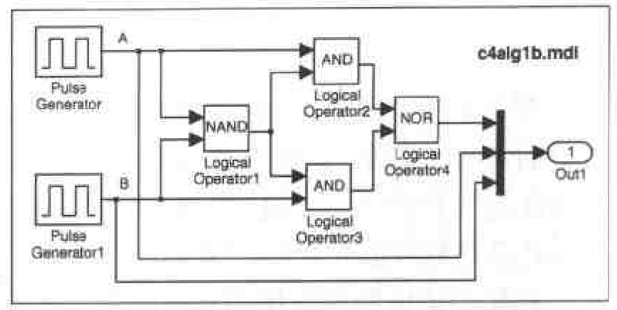
即用 $(s+3)/(2s+4)$ 取代原来的反馈回路，构造出如图 4-48 (b) 所示的系统框图，这样既能正确地计算出系统的响应，又能成功地避免代数环错误。如果用控制系统工具箱中的线性对象模块，则可以构造出如图 4-48 (c) 所示的模型，该模型也能起到相同的作用。

〖例 4.6〗 Simulink 的数学函数模块组中提供了"逻辑算子" (Logic operator) 模块，可以搭建数

字逻辑电路。双击该模块，则得出如图 4-49 (a) 所示的对话框，在其 Logic (编辑运算) 栏目中可以选择各种各样的逻辑运算关系。



(a) 逻辑模块对话框



(b) Simulink 模型

图 4-49 逻辑电路元件对话框与仿真框图

考虑下面的逻辑关系式

$$Z = \overline{A \cdot \overline{A \cdot B} + B \cdot \overline{A \cdot B}} \tag{4.6}$$

其中的 $\overline{A \cdot B}$ 用 "与非门" (NAND) 即可表示，所以可以用 Simulink 中提供的逻辑运算符号来搭建出如图 4-49 (b) 所示的数字逻辑电路图。在两路给定的输入信号中，A 路信号直接采用脉冲信号，B 路信号亦采用脉冲模块，但将其延迟时间 设置为 0.5。

在仿真中选择定步长算法，并设步长为 0.01，则可以进行仿真，仿真结束后，可以给出如下命令，就能得出如图 4-50 所示的仿真结果。

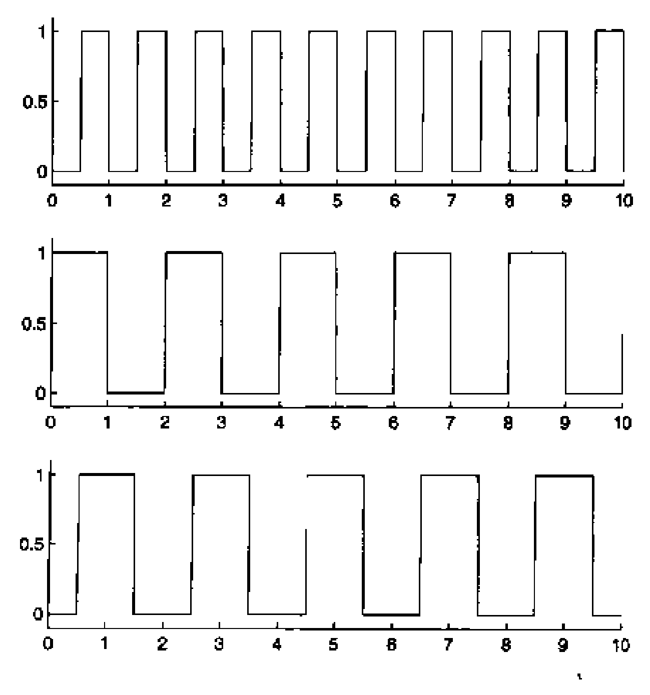


图 4-50 逻辑电路的仿真效果

```
>> subplot(311), plot(tout,yout(:,1)), set(gca,'ylim',[-0.1,1.1],'Box','off')
   subplot(312), plot(tout,yout(:,2)), set(gca,'ylim',[-0.1,1.1],'Box','off')
   subplot(313), plot(tout,yout(:,3)), set(gca,'ylim',[-0.1,1.1],'Box','off')
```

4.4　线性系统的计算机仿真

假设系统对输入 $u_1(t)$ 信号的响应为 $y_1(t)$，而对 $u_2(t)$ 输入信号的响应为 $y_2(t)$，若对任意的常数 a 和 b，系统对输入信号 $au_1(t)+bu_2(t)$ 的响应可以表示成 $ay_1(t)+by_2(t)$，则称系统是线性的。这一性质又称为线性系统的叠加原理。换句话说，所有满足叠加原理的系统都是线性的。显然，例 4.3 中给出的模型显然不满足叠加原理。

控制理论中经常使用的传递函数和零极点模型都是线性模型，而形如式 (4.1) 描述的状态方程也是线性模型。由于线性系统有自己的特性，分析起来比一般非线性系统容易得多，另外还可以采用间接的方式，如频域方法来分析系统的时域性质。

4.4.1　线性系统的数学模型

在 MATLAB 的控制系统工具箱中定义了几个常用的线性模型对象，如 tf() 可以表示传递函数模型，ss() 表示状态方程模型，而 zpk() 表示零极点模型。

〖例 4.7〗 假设已知系统的传递函数模型为：

$$G(s)=\frac{s^3+7s^2+24s+24}{s^4+10s^3+35s^2+50s+24}$$

由下面的 MATLAB 语句可以容易地得出系统输入到工作空间中：

```
>> num=[1,7,24,24]; den=[1,10,35,50,24]; G=tf(num,den)
Transfer function:
      s^3 + 7 s^2 + 24 s + 24
 ---------------------------------
  s^4 + 10 s^3 + 35 s^2 + 50 s + 24
```

这样就可以用单一的变量名 G 来描述系统的数学模型了。不同模型的相互转换也是很容易的，例如可以用下面的命令直接获得系统的零极点模型。

```
>> G1=zpk(G)  % 显示系统的零极点模型
 Zero/pole/gain:
 (s+1.539) (s^2 + 5.461s + 15.6)
 -------------------------------
    (s+4) (s+3) (s+2) (s+1)
```

已知线性系统的数学模型后，可以容易地得出简单互连模型的总模型，例如两个串联的模型分别为 G1 和 G2，则总的模型为 G=G1*G2。并联模型和反馈模型分别由 G=G1+G2 和 feedback(G1,G2) 求出。

〖例 4.8〗 比较复杂的系统模型也可以利用这些简单的命令求出，例如例 4.2 中的直流拖动系统 模型可以由下面语句直接求出：

```
>> g1=tf(1,[0.01,1]); g2=tf([0.17,1],[0.085,0]); g3=g1;
   g4=tf([0.15,1],[0.051,0]); g5=tf(70,[0.0067,1]); g6=tf(0.21,[0.15,1]);
   g7=tf(130,[1,0]); g8=0.212; g9=tf(0.1,[0.01,1])*inv(g7); g10=0.0044*g1;
   gg1=feedback(g7*g6,g8); gg2=feedback(gg1*g5*g4,g9);
   G=feedback(gg2*g3*g2,g10)*g1; zpk(G) %总系统模型的零极点显示
Zero/pole/gain:
           111852502194.908 (s+100)^2 (s+6.667) (s+5.882)
-----------------------------------------------------------------------
(s+180.9)(s+100)^2(s+84.12)(s+48.21)(s^2+15.16s+74.33)(s^2+27.57s+354)
```

观察上述结果不难发现，分子和分母同时含有 $(s+100)^2$ 项，说明这样得出的模型仍然可以进一步化简。利用控制系统工具箱中提供的 minreal() 获得系统的最小实现模型

```
>> G=zpk(minreal(G))
Zero/pole/gain:
                111852502194.908 (s+6.667) (s+5.882)
---------------------------------------------------------------------------
(s+180.9) (s+48.21) (s+84.12) (s^2 + 15.16s + 74.33) (s^2 + 27.57s + 354)
```

4.4.2 线性连续系统的解析解

假设线性系统由传递函数给出：

$$G(s)=\frac{b_1s^m+b_2s^{m-1}+\cdots+b_ms+b_{m+1}}{s^n+a_1s^{n-1}+a_2s^{n-2}+\cdots+a_{n-1}s+a_n} \tag{4.7}$$

如果 $G(s)$ 含有 n 个不同的极点 $p_i\ (i=1,2,\cdots,n)$，则该传递函数的部分分式展开可以写成：

$$G(s)=\frac{r_1}{s-p_1}+\frac{r_2}{s-p_2}+\cdots+\frac{r_n}{s-p_n} \tag{4.8}$$

式中 r_i 与 p_i 既可以为实数，又可以为复数。因为一阶传递函数 $G_1(s)=b/(s+a)$ 的 Laplace 逆变换为 $\mathcal{L}^{-1}[G_1(s)]=b\mathrm{e}^{-at}$，所以系统时域响应可以通过对上式进行 Laplace 逆变换得出

$$g(t)=\mathcal{L}^{-1}[G(s)]=r_1\mathrm{e}^{p_1t}+r_2\mathrm{e}^{p_2t}+\cdots+r_n\mathrm{e}^{p_nt} \tag{4.9}$$

如果 $G(s)$ 模型的第 j 个极点 p_j 是 m 重的，则在部分分式展开中将含有下面各项

$$\frac{r_j}{s-p_j}+\frac{r_{j+1}}{(s-p_j)^2}+\cdots+\frac{r_{j+m-1}}{(s-p_j)^m} \tag{4.10}$$

该形式对应的 Laplace 逆变换为

$$r_j\mathrm{e}^{p_jt}+r_{j+1}t\mathrm{e}^{p_jt}+\cdots+\frac{r_{j+m-1}}{(m-1)!}t^{m-1}\mathrm{e}^{p_jt}=\left[r_j+r_{j+1}t+\cdots+\frac{r_{j+m-1}}{(m-1)!}t^{m-1}\right]\mathrm{e}^{p_jt} \tag{4.11}$$

可见，线性系统的脉冲响应解析解可以通过部分分式展开技术直接得出。如果系统的输入信号可以由一般的 Laplace 式子 $R(s)$ 给出，则同样能将输出信号 $Y(s)=G(s)R(s)$ 的 Laplace 逆变化求出。

MATLAB 语言中提供了一个 residue() 函数来对有理传递函数 (num, den) 进行部分分式展开，该函数的调用格式为：

```
[R,P,K]=residue(num,den)
```

其中 (R, P) 分别为各个子传递函数的增益和极点位置，而 K 为部分分式展开后的余项，若系统模型为非严格正则的，即

$$Y(s)=\frac{\mathbf{num}(s)}{\mathbf{den}(s)}=\frac{b_0s^n+b_1s^{n-1}+\cdots+b_{n-1}s+b_n}{s^n+a_1s^{n-1}a_{n-2}s^{n-2}+\cdots+a_{n-1}s+a_n} \tag{4.12}$$

则余项 K 应该返回 b_0。

〖例 4.9〗 考虑一个传递函数模型：

$$G(s)=\frac{s^3+7s^2+24s+24}{s^4+10s^3+35s^2+50s+24}$$

由下面的 MATLAB 语句可以容易地得出系统的阶跃响应解析解

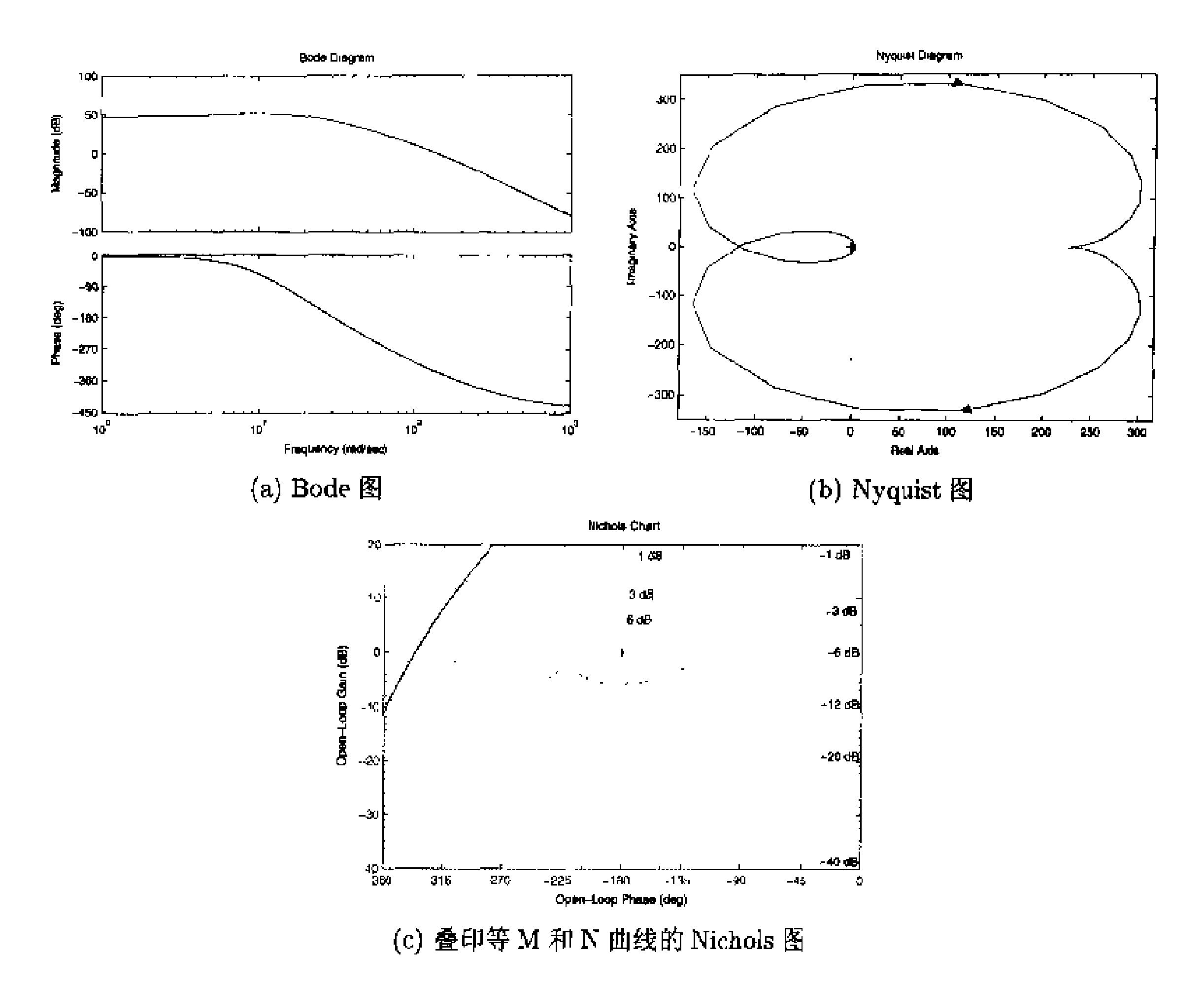
```
>> num=[1,7,24,24]; den=[1,10,35,50,24];
   [r,p,k]=residue(num,[den,0]); [r,p], k
ans =
     -1.0000    -4.0000
      2.0000    -3.0000
     -1.0000    -2.0000
     -1.0000    -1.0000
      1.0000          0
k =
          []
```

该解所表示的数学式子为 $y(t)=-\mathrm{e}^{-4t}+2\mathrm{e}^{-3t}-\mathrm{e}^{-2t}-\mathrm{e}^{-t}+1$。因为阶跃输入的 Laplace 变换为 $U(s)=1/s$，故输出信号的 Laplace 变换可以写成 $Y(s)=G(s)/s$，这样在 MATLAB 语言下，可以很自然地用 num，[den,0] 来表示 $Y(s)$ 信号的分子和分母了。

4.4.3 线性系统频域分析

MATLAB 的控制系统工具箱中提高了诸多系统频域分析，如系统的 Bode 图可以由 bode() 函数直接绘制出来，而系统的 Nyquist 图可以由 nyquist() 函数直接绘制出来，Nichols 图可以由 nichols() 函数绘制出来，如果需要叠印等 M 线和等 N 线，则可以使用 ngrid() 函数[53]。

〖例 4.10〗 例 4.2 中模型的 Bode 图、Nyquist 图和叠印等 M 和等 N 线的 Nichols 图可以分别由下面的命令绘制出来，分别如图 4-51 所示。



(a) Bode 图

(b) Nyquist 图

(c) 叠印等 M 和 N 曲线的 Nichols 图

图 4-51 系统的频域响应曲线

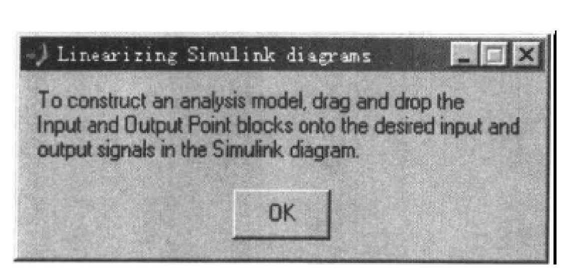
```
>> g1=tf(1,[0.01,1]); g2=tf([0.17,1],[0.085,0]); g3=g1;
   g4=tf([0.15,1],[0.051,0]); g5=tf(70,[0.0067,1]); g6=tf(0.21,[0.15,1]);
   g7=tf(130,[1,0]); g8=0.212; g9=tf(0.1,[0.01,1])*inv(g7); g10=0.0044*g1;
   gg1=feedback(g7*g6,g8); gg2=feedback(gg1*g5*g4,g9);
   G=feedback(gg2*g3*g2,g10)*g1; % 总系统模型
   bode(G); % 绘制 Bode 图
   figure; nyquist(G) % 绘制 Nyquist 图
   figure; ngrid('new'); % 绘制等 M 和等 N 曲线
   nichols(G), axis([-360,0,-40,20]); % 叠印 Nichols 图
```

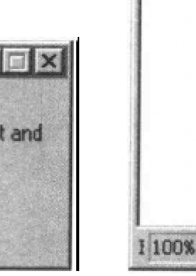
当然这样得出的频域响应曲线是针对闭环系统的，不是传统意义下针对开环系统的，因为在语句调用中使用的是闭环模型。

4.4.4 Simulink 下的线性系统分析工具

在 Simulink 下也提供了线性系统分析的工具，选中 Tools | Linear System 菜单项，则将打开线性系统分析工具，用户可以选择其中的 Get Linear Model 菜单项来提取系统的线

性模型，这时 Simulink 将自动给出一个如图 4-52 (a) 所示的信息框，并同时将打开一个





(a) 提示框 (b) 输入输出端口库

图 4-52 系统的频域响应曲线

如图 4-52 (b) 所示的的模型库窗口，提示用户给要分析的 Simulink 模型指定输入和输出端口，指定了端口的直流电机拖动系统 Simulink 框图 如图 4-53 所示。提取了系统的线性

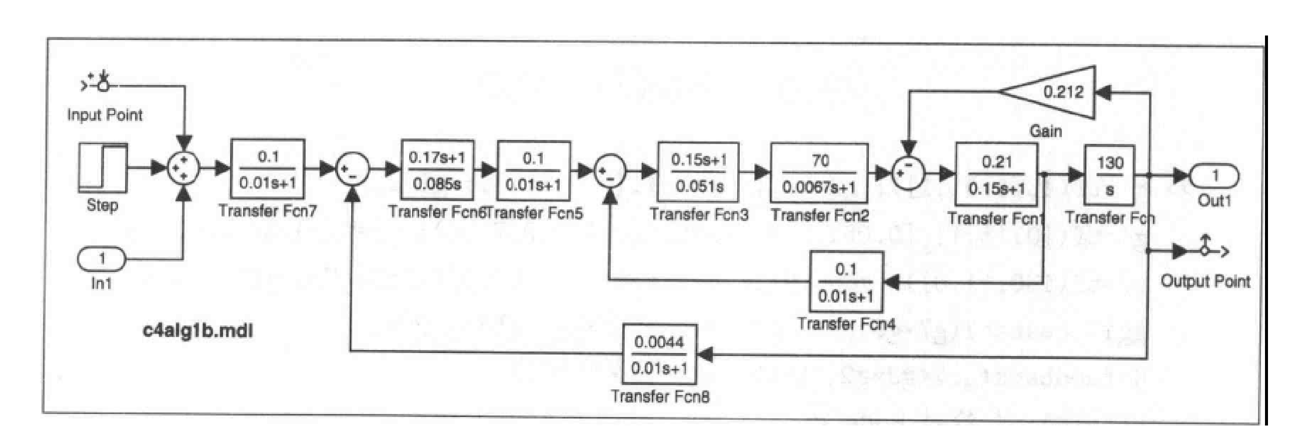


图 4-53 添加输入输出端口的电机拖动模型

模型，则可以在线性系统分析窗口中直接分析系统的性能，如系统的阶跃响应可以直接得出，如图 4-54 所示。

选择该窗口中的 Edit | Plot Configuration 菜单项，则将得出如图 4-55 所示的对话框，从该对话框下可以选择系统分析的任务与图形布置。例如，在默认的布局下，只显示系统的阶跃响应曲线，用户还可以适当地选择该对话框中其他图形布局形式，在一个图形窗口中同时绘制出若干种图形。例如选择第 2 种形式，将得出如图 4-56 所示的显示效果，其中图形的上部分为系统的阶跃响应曲线，下部分为系统的 Nyquist 图。

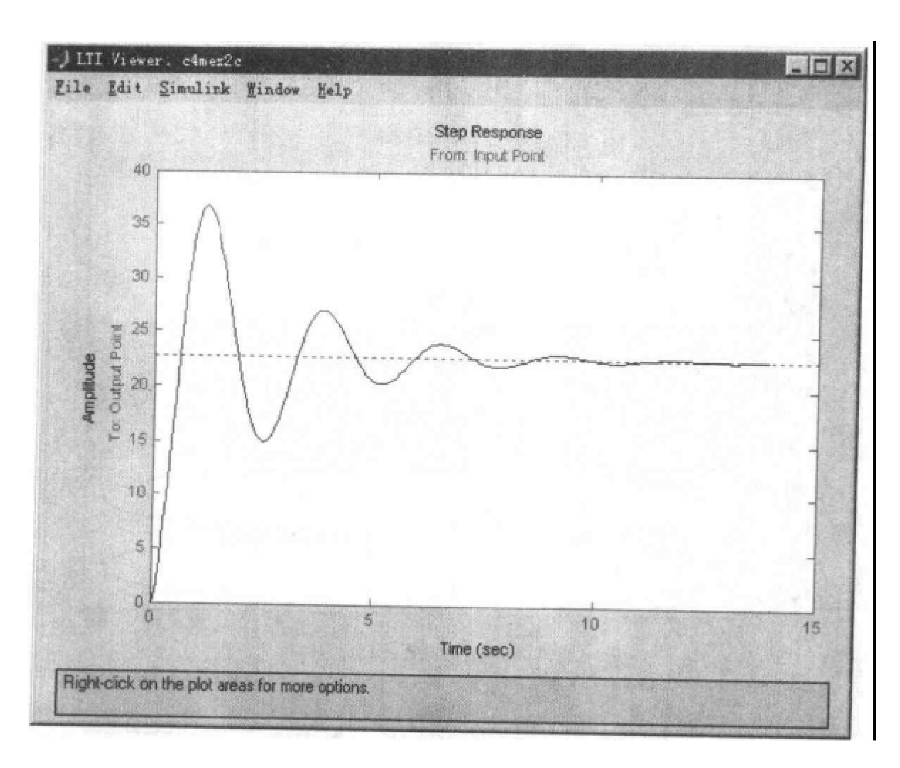


图 4-54 系统的阶跃响应曲线

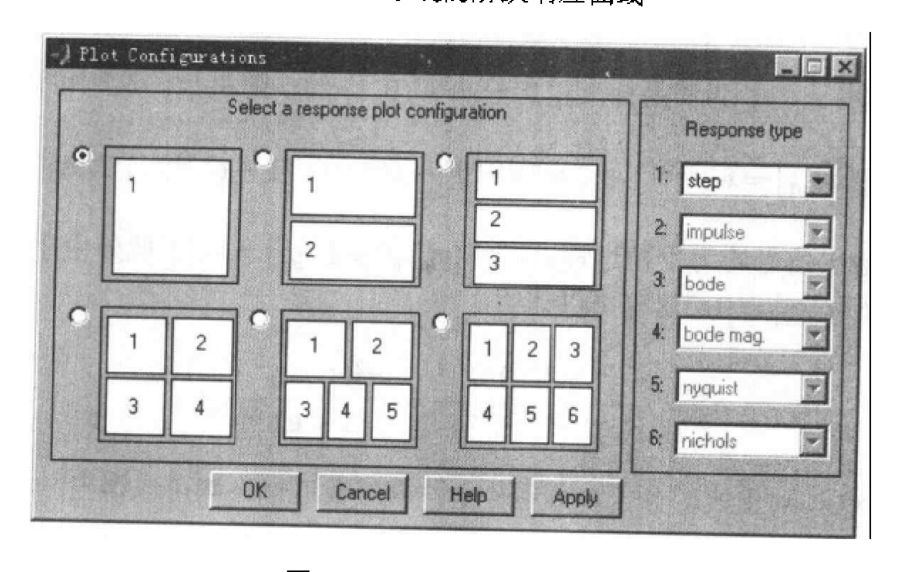


图 4-55 图形设置对话框

4.5 随机输入下连续系统仿真

考虑下面给出的一阶线性模型 $G(s) = 1/(as + 1)$, a 为给定常数。假设输入为 Gauss 白噪声信号 $\gamma(t)$，其均值为 0，方差为 σ^2，可以证明[2]，输出函数 $y(t)$ 亦为 Gauss 信号，其均值为 0，方差为 $\sigma_y^2 = \sigma^2/(2a)$。如果假设在一个计算步长内输入信号为一常数 e_k，并直接对此系统进行离散化，则可以得出系统的模型为：

$$y_{k+1} = \mathrm{e}^{-\Delta t/a} y_k + (1 - \mathrm{e}^{-\Delta t/a})\sigma e_k \tag{4.13}$$

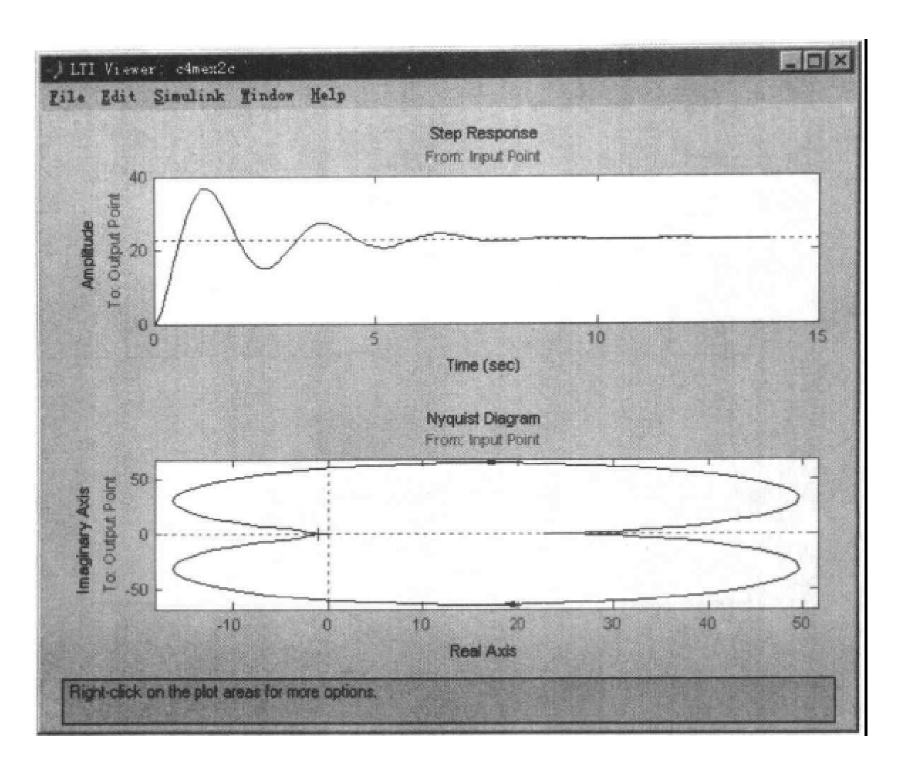


图 4-56 同时显示多种系统响应

式中 Δt 为计算步长，e_k 为满足标准正态分布 $N(0,1)$ 的伪随机数。这时可以得出

$$E[y_{k+1}^2]=\mathrm{e}^{-2\Delta t/a}+2\sigma\mathrm{e}^{-\Delta t/a}E[e_ky_k]+\sigma^2(1-\mathrm{e}^{-\Delta t/a})^2E[e_k^2] \tag{4.14}$$

若输入与输出信号均为平稳过程，则 $E[y_{k+1}^2]=E[y_k^2]=\sigma_y^2$，此外由于 y_k 与 e_k 是相互独立的，则 $E[y_ke_k]=0$。这时可以证明

$$\sigma_y^2=\frac{\sigma^2(1-\mathrm{e}^{-\Delta t/a})^2}{(1-\mathrm{e}^{-2\Delta t/a})}=\frac{\sigma^2(1-\mathrm{e}^{-\Delta t/a})}{1+\mathrm{e}^{-\Delta t/a}} \tag{4.15}$$

若 $\Delta t/a\to 0$，对上式的分子和分母分别作幂级数近似，则可以得出：

$$\sigma_y^2=\lim_{\Delta t/a\to 0}\frac{\Delta t/a+o[(\Delta t/a)^2]}{2+o(\Delta t/a)}\sigma^2=\frac{\Delta t}{2a}\sigma^2 \tag{4.16}$$

可见，这样得出的输出函数的方差取决于计算步长 Δt，这一结果是不正确的。由此可见，如果输入函数为随机信号时，不能采用传统的方法进行仿真。

鉴于此问题，在一些仿真软件中建议使用其他随机过程来近似替代原始的 Gauss 白噪声信号，例如在一个著名的仿真语言 ACSL[34] 中建议使用 Ornstein-Uhlenbeck 过程来近似地替代原始的白噪声信号，其目的是要在一个指定的频率范围内保持一个恒定的输入功率。Ornstein-Uhlenbeck 过程定义为：

$$n_{k+1}=n_k\mathrm{e}^{-\Delta t/T_f}+\sigma\sqrt{1-\mathrm{e}^{-2\Delta t/T_f}}\,e_k \tag{4.17}$$

式中 T_f 为相关时间常数，Δt 为计算步长，e_k 为满足标准正态分布 $N(0,1)$ 的伪随机数。可以证明，Ornstein-Uhlenbeck 过程的相关函数满足：

$$R(\tau)=\sigma^2\mathrm{e}^{-|\tau|/T_f},\ \ -\infty<\tau<\infty \tag{4.18}$$

若使用 Ornstein-Uhlenbeck 过程来替代 Gauss 白噪声输入，则可以得出仿真结果以及假设检验如表 4-1 所示[50, 55]。可以看出，在一般的仿真步长下用传统的仿真方法所得出的仿真结果仍是不正确的。

表 4-1 ACSL 仿真结果及假设检验

T_f	Δt	均值 $\bar{y}$	方差 $\hat{\sigma}^2$	均值假设检验 $\eta_{\bar{y}}$	方差假设检验 $\eta_{\hat{\sigma}^2}$	正态性假设检验 η_N
0.02	0.02	-0.0116	0.0650	-0.9443	-55.9919	9.6715
0.02	0.04	-0.0091	0.0800	-0.7417	-54.3777	2.5956
0.1	0.02	-0.0250	0.2788	-2.0453	-40.2281	19.1610
0.1	0.04	-0.0180	0.2852	-1.4692	-39.8802	5.2735
0.5	0.02	-0.0551	1.0090	-4.4951	-12.7166	62.2528
0.5	0.04	-0.0409	1.0242	-3.3389	-12.2809	27.2177
2	0.02	-0.0979	2.0283	-7.9923	11.5146	57.6419
2	0.04	-0.0778	2.0177	-6.3567	11.2994	62.0466

4.5.1 线性系统的仿真研究

假设线性连续系统的状态方程模型为：

$$\dot{\boldsymbol{x}}(t)=\boldsymbol{A}\boldsymbol{x}(t)+\boldsymbol{B}[d(t)+\gamma(t)],\ \ y(t)=\boldsymbol{C}\boldsymbol{x}(t) \tag{4.19}$$

式中 $\boldsymbol{A}\in\mathbb{R}^{n\times n}$, $\boldsymbol{B}\in\mathbb{R}^{n\times m}$, $\boldsymbol{C}\in\mathbb{R}^{r\times n}$, $d(t)\in\mathbb{R}^{m\times 1}$ 为确定性输入向量，$\gamma(t)\in\mathbb{R}^{m\times 2}$ 为 Gauss 白噪声向量，满足下式：

$$E[\gamma(t)]=0,\ \ E[\gamma(t)\gamma^T(t)]=\boldsymbol{V}_\sigma\delta(t-\tau) \tag{4.20}$$

定义一个变量 $\gamma_c(t)=\boldsymbol{B}\gamma(t)$，则可以证明 $\gamma_c(t)$ 亦为 Gauss 白噪声，满足

$$E[\gamma_c(t)]=0,\ \ E[\gamma_c(t)\gamma_c^T(t)]=\boldsymbol{V}_c\delta(t-\tau) \tag{4.21}$$

其中 $\boldsymbol{v}_c=\boldsymbol{B}\boldsymbol{V}\sigma\boldsymbol{B}^T\in\mathbb{R}^{m\times m}$ 为一个协方差矩阵，这时式 (4.19) 可以改写成

$$\dot{\boldsymbol{x}}(t)=\boldsymbol{A}\boldsymbol{x}(t)+\boldsymbol{B}d(t)+\gamma_c(t),\ \ y(t)=\boldsymbol{C}\boldsymbol{x}(t) \tag{4.22}$$

状态变量的解析解可以写成：

$$\boldsymbol{x}(t)=\mathrm{e}^{-\boldsymbol{A}t}\boldsymbol{x}(t_0)+\int_{t_0}^{t}\mathrm{e}^{\boldsymbol{A}(t-\tau)}d(\tau)\boldsymbol{B}\mathrm{d}\tau+\int_{t_0}^{t}\gamma_c(t)\mathrm{d}\tau \tag{4.23}$$

假设 $t_0 = k\Delta t, t = (k+1)\Delta t$，其中 Δt 为计算步长，并假定在一个计算步长之内确定性输入信号 $d(t)$ 为一个常数，亦即，如 $\Delta t \le t \le (k+1)\Delta t$ 时有 $d(t) = d(k\Delta t)$，则式 (4.23) 的离散形式可以写成

$$\boldsymbol{x}[(k+1)\Delta t] = \boldsymbol{F}\boldsymbol{x}(k\Delta t) + \boldsymbol{G}d(k\Delta t) + \gamma_d(k\Delta t),\ \ y(k\Delta t) = \boldsymbol{C}\boldsymbol{x}(k\Delta t) \tag{4.24}$$

式中 $\boldsymbol{F} = \mathrm{e}^{\boldsymbol{A}\Delta t}$, $\boldsymbol{G} = \int_0^{\Delta t} \mathrm{e}^{\boldsymbol{A}(\Delta t - \tau)}\boldsymbol{B}\mathrm{d}\tau$，且

$$\gamma_d(k\Delta t) = \int_{k\Delta t}^{(k+1)\Delta t} \mathrm{e}^{\boldsymbol{A}[(k+1)\Delta t - \tau]}\gamma_c(t)\mathrm{d}\tau = \int_0^{\Delta t} \mathrm{e}^{\boldsymbol{A}t}\gamma_c[(k+1)\Delta t - \tau]\mathrm{d}\tau \tag{4.25}$$

可见矩阵 $\boldsymbol{F}, \boldsymbol{G}$ 和确定性系统的离散化形式是一样的，所以会很容易地求得，但可以看出，若系统含有随机输入时，系统的离散化形式与传统形式是不同的。可以证明 $\gamma_d(t)$ 亦为 Gauss 白噪声向量，且满足：

$$E[\gamma_d(k\Delta t)] = 0,\ \ E[\gamma_d(k\Delta t)\gamma_d^T(j\Delta t)] = \boldsymbol{V}\delta_{kj} \tag{4.26}$$

式中 $\boldsymbol{V} = \int_0^{\Delta t} \mathrm{e}^{\boldsymbol{A}t}\boldsymbol{V}_c\mathrm{e}^{\boldsymbol{A}^T t}\mathrm{d}t$。利用 Taylor 级数展开技术可得出：

$$\boldsymbol{V} = \int_0^{\Delta t} \sum_{k=0}^{\infty} \frac{\boldsymbol{R}^{(k)}(0)}{k!} t^k \mathrm{d}t = \sum_{k=0}^{\infty} \boldsymbol{V}_k \tag{4.27}$$

其中 $\boldsymbol{R}^{(k)}(0)$ 与 $\boldsymbol{V}_k$ 可以由下式递推地求出：

$$\begin{cases} \boldsymbol{R}^{(k)}(0) = \boldsymbol{A}\boldsymbol{R}^{(k-1)}(0) + \boldsymbol{R}^{(k-1)}(0)\boldsymbol{A}^T \\ \boldsymbol{V}_k = \dfrac{\Delta t}{k+1}(\boldsymbol{A}\boldsymbol{V}_{k-1} + \boldsymbol{V}_{k-1}\boldsymbol{A}^T) \end{cases} \tag{4.28}$$

递推初值为 $\boldsymbol{R}^{(0)}(0) = \boldsymbol{R}(0) = \boldsymbol{V}_c$, $\boldsymbol{V}_0 = \boldsymbol{V}_c\Delta t$。由奇异值分解理论，可以将矩阵 $\boldsymbol{V}$ 写成 $\boldsymbol{V} = \boldsymbol{U}\boldsymbol{\Gamma}\boldsymbol{U}^T$，其中 $\boldsymbol{U}$ 为正交矩阵，$\boldsymbol{\Gamma}$ 为含有非零对角元素的对角矩阵，这样可以得出 Cholesky 分解 $\boldsymbol{V} = \boldsymbol{D}\boldsymbol{D}^T$。且 $\gamma_d(k\Delta t) = \boldsymbol{D}\boldsymbol{e}(k\Delta t)$，式中 $\boldsymbol{e}(k\Delta t) \in \mathbb{R}^{n\times 1}$，且 $\boldsymbol{e}(k\Delta t) = (e_k, e_{k+1}, \cdots, e_{k+n-1})^T$，使得各个分量 e_k 满足标准正态分布，即 $e_k \sim N(0,1)$。系统的离散形式的递推解可以写成

$$\boldsymbol{x}[(k+1)\Delta t] = \boldsymbol{F}\boldsymbol{x}(k\Delta t) + \boldsymbol{G}d(k\Delta t) + \boldsymbol{D}\boldsymbol{e}(k\Delta t),\ \ y(k\Delta t) = \boldsymbol{C}\boldsymbol{x}(k\Delta t) \tag{4.29}$$

根据上面的算法，可以编写出随机输入下连续线性系统离散化的 MATLAB 函数 `sc2d()` 如下：

```
function [F,G,D,C]=sc2d(G,V,T)
G=ss(G); G=balreal(G); A=G.a; B=G.b; C=G.c;
[F,G]=c2d(A,B,T); V0=B*V*B'*T; Vd=V0;
vmax=sum(sum(abs(Vd))); vv=vmax; i=1;
while (1)
```

```
        V1 = T/(i+1)*(A*V0+V0*A'); v0 = sum(sum(abs(V1)));
        Vd = Vd+V1; V0 = V1; vv = [vv v0]; i=i+1;
        if v0 < 1e-10*vmax, break; end
    end
    [U,S,V0]=svd(Vd); V0=sqrt(diag(S));
    Vd=diag(V0); D=U*Vd;
```

在仿真时，可以产生一组伪随机数，从而产生向量 $\boldsymbol{e}(k\Delta t)$，然后求出状态变量 $\boldsymbol{x}[(k+1)\Delta t]$ 并求出输出变量 $y[(k+1)\Delta t]$。重新考虑一下前面给出的一阶系统的例子，系统的状态方程可以写成：

$$\dot{y}(t) = -\frac{1}{a}y(t) + \frac{1}{a}\gamma_0(t) \tag{4.30}$$

系统的输出信号 $y(t)$ 的离散形式可以写成：

$$y_{k+1} = \mathrm{e}^{\Delta t/a}y_k + \sigma\sqrt{\frac{1}{2a}\left(1-\mathrm{e}^{-2\Delta t/a}\right)}e_k \tag{4.31}$$

可见系统的解和前面得出的用 Ornstein-Uhlenbeck 模拟得出的结果是不同的，因为 Ornstein-Uhlenbeck 过程是用来模拟白噪声输入信号的。

仿照前面的方法可以证明 $\sigma_y^2 = \sigma^2/(2a)$，这一方差与理论值是一致的。此外，和前面所用的传统方法不同，用此方法得出的仿真结果的统计量与仿真步长的选取无关。应用此仿真算法也可以对不同的仿真步长得出其均值与方差及其假设检验统计量，如表 4-2 所示。由表 4-2 可以看出，本仿真方法得出的结果是可信的。

表 4-2 仿真结果的统计分析与假设检验

步长 Δ	均值 $\bar{y}$	方差 $\hat{\sigma}^2$	均值假设检验 $\eta_{\bar{y}}$	方差假设检验 $\eta_{\hat{\sigma}^2}$	正态性假设检验 η_N
0.02	-2.0753×10^{-6}	1.4305	-2.9350×10^{-4}	-5.6745	7.7593
0.05	-5.5590×10^{-4}	1.4338	0.0786	-5.4015	2.0494
0.10	6.3233×10^{-4}	1.4626	0.0894	-3.0512	0.9768
0.20	4.0403×10^{-4}	1.4913	0.0571	-0.7089	0.5862
0.50	1.4048×10^{-4}	1.5058	0.0199	0.4699	-1.7340

【例 4.11】 考虑例 4.7 中所示的系统模型，如果用白噪声信号激励该系统，并假设系统的采样周期为 $T=0.001$，则得出的状态方程模型可以由下面的 MATLAB 语句得出

```
>> G=tf([1,7,24,24],[1,10,35,50,24]); T=0.02;
   [F,G0,D,C]=sc2d(G,1,T)
F =

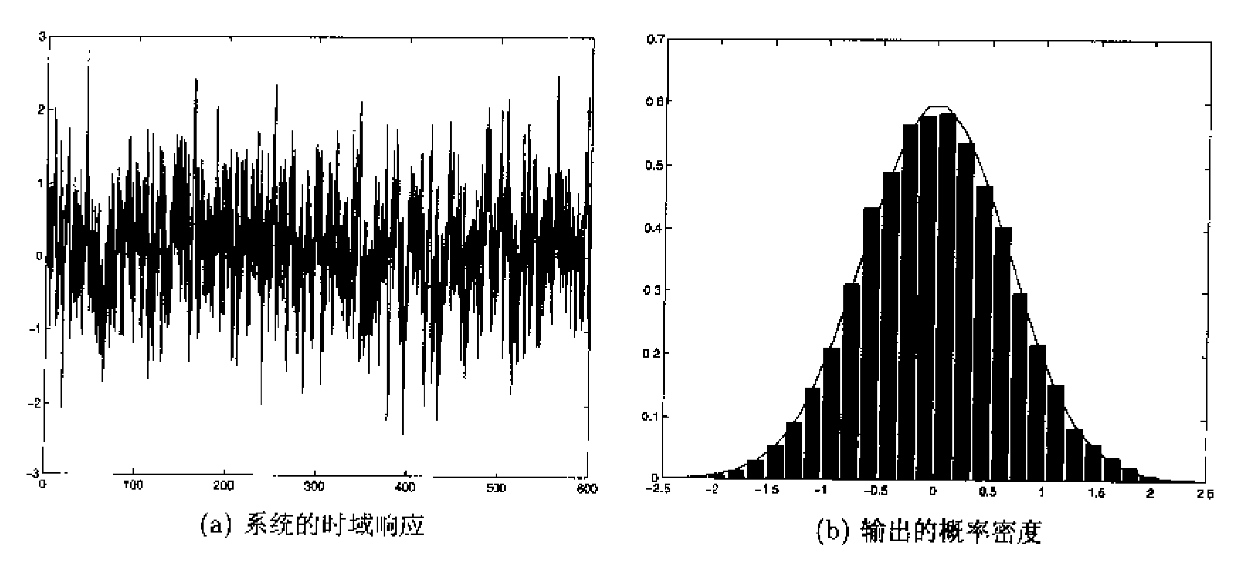
    0.9838    0.0067    0.0132   -0.0013
   -0.0067    0.9883   -0.0702    0.0036
    0.0132    0.0702    0.8653    0.0257
```

```
   -0.0013   -0.0036    0.0257    0.9684
GO =
   -0.0182
   -0.0036
    0.0076
   -0.0007
D =
   -0.1289   -0.0009   -0.0000   -0.0000
   -0.0251   -0.0012    0.0000    0.0000
    0.0536   -0.0026   -0.0000    0.0000
   -0.0051    0.0005   -0.0000    0.0000
C =
   -0.9216    0.1663    0.4201   -0.0431
```

由仿真模型 (即得出的差分方程) 出发，可以用下列的 MATLAB 语句对之进行仿真，其中仿真点数设为 30000个。

```
>> n_point=30000; r=randn(n_point+4,1); r=r-mean(r);
   y=zeros(n_point,1); x=zeros(4,1); d0=0;
   for i=1:n_point
      x=F*x+G0*d0+D*r(i:i+3); y(i)=C*x;
   end
   t=0:.02:(n_point-1)*0.02; plot(t,y)
```

得出的响应曲线如图 4-57(a) 所示，不过从得出的曲线看，这样的响应似乎杂乱无章，所以对随机输入来说，分析其统计规律应该更有用。



(a) 系统的时域响应　　(b) 输出的概率密度

图 4-57　随机输入系统的响应

在白噪声激励下系统输出的方差可以由控制系统工具箱中的 covar() 函数求出，该函数的调用格式为：

```
v=covar(G, Sig)
```

其中 G 为系统模型，Sig 为输入白噪声的方差，得出的 v 为输出信号的方差。如果系统为多输出的，则 v 为输出信号的协方差。

得出了输出信号和方差的理论值，则可以用下面的语句得出系统输出的概率密度曲线与仿真结果，如图 4-57 (b) 所示，可以看出这样的仿真算法结果是可信的。

```
>> v=covar(G,1); [v, cov(y)]
ans =
    0.4429    0.4447
>> xx=linspace(-2.5,2.5,30); yy=hist(y,xx);
   yy=yy/(30000*(xx(2)-xx(1))); yp=exp(-xx.^2/(2*v))/sqrt(2*pi*v);
   bar(xx,yy), hold on; plot(xx,yp)
```

4.5.2 在 Simulink 下的解决方法

由前面的叙述可知，上节中给出的仿真方法只可以用于线性系统的处理，而不能直接用于非线性系统，尽管对某一类非线性系统利用前面给出的方法可以推导出仿真的近似解[55]，但很难得出一般非线性系统的仿真解法。在这一节中将讨论另一种有效的近似方法。为了得出系统的近似解，必须使用定步长的仿真方法，并对伪随机输入函数进行比例化。假定比例化系数为 K_σ，即 $\gamma_i = K_\sigma e_i$，式中 e_i 为满足正态分布的伪随机数，而 γ_i 为可以用于仿真的实际输入量。假定在一个计算步长内输入信号 γ_i 为一常值，即

$$\gamma(t) = \gamma_i = K_\sigma e_i,\ \ i\Delta t \leqslant t < (i+1)\Delta t \tag{4.32}$$

为保证伪随机变量 γ_i 的强度与原随机信号 $\gamma_0(t)$ 相同，则可以证明[55]：

$$K_\sigma = \sigma\sqrt{\frac{1}{\Delta t}} \tag{4.33}$$

考虑前面的一阶系统的例子，对输入函数进行加权后，系统的仿真模型可以写成

$$y_{k+1} = \mathrm{e}^{-\Delta t/a}y_k + \sigma\sqrt{\frac{1}{\Delta t}}(1-\mathrm{e}^{-\Delta t/a})e_k$$

比较一下前面给出的例子，不难看出当 Δt 很小时：

$$\frac{\sigma\sqrt{\dfrac{1}{\Delta t}}(1-\mathrm{e}^{-\Delta t/a})}{\sigma\sqrt{\dfrac{1}{2a}}(1-\mathrm{e}^{-2\Delta t/a})} = \sqrt{\frac{2a(1-\mathrm{e}^{-\Delta t/a})}{\Delta t(1+\mathrm{e}^{-\Delta t/a})}} = \sqrt{\frac{2a[\Delta t/a+o(\Delta t^2)]}{\Delta t[2+o(\Delta t)]}} \xrightarrow{\Delta t\to 0} 1$$

即这一方法在 Δt 很小时得出的统计结果是很接近理论值的的。采用此仿真方法对不同的步长所得出的仿真结果的均值和方差以及假设检验的结果如表 4-3 所示。可见，在步长不是很大时所得出的结果是可信的。

表 4-3 仿真结果的统计分析与假设检验

步长 Δ	均值 $\bar{y}$	方差 $\hat{\sigma}^2$	均值假设检验 $\eta_{\bar{y}}$	方差假设检验 $\eta_{\hat{\sigma}^2}$	正态性假设检验 η_N
0.02	8.4025×10^{-8}	1.4448	1.1883×10^{-6}	-4.5100	6.7938
0.05	4.0632×10^{-4}	1.4711	0.0575	-2.3605	1.2797
0.10	4.5481×10^{-4}	1.5417	0.0643	3.4008	0.4793
0.20	2.4612×10^{-4}	1.6601	0.0348	13.0691	0.3849
0.50	4.9850×10^{-4}	2.0072	0.0071	41.4094	0.5757

〖例 4.12〗 再考虑例 4.7 中给出的系统模型，根据给出的模型，应用本节给出的近似仿真算法，则可以得出如图 4-58 所示的 Simulink 模型，选择 T=0.1， 并调用菜单项 Simulation | Simulation

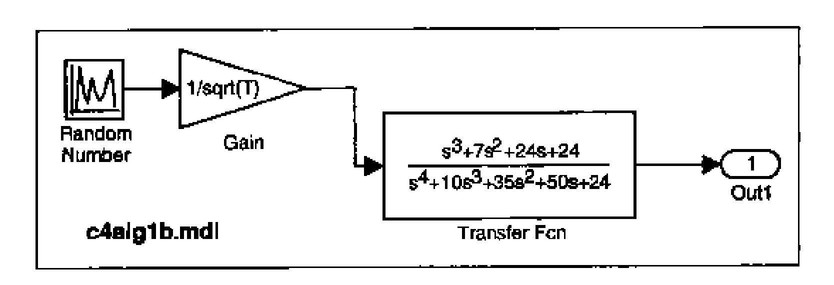
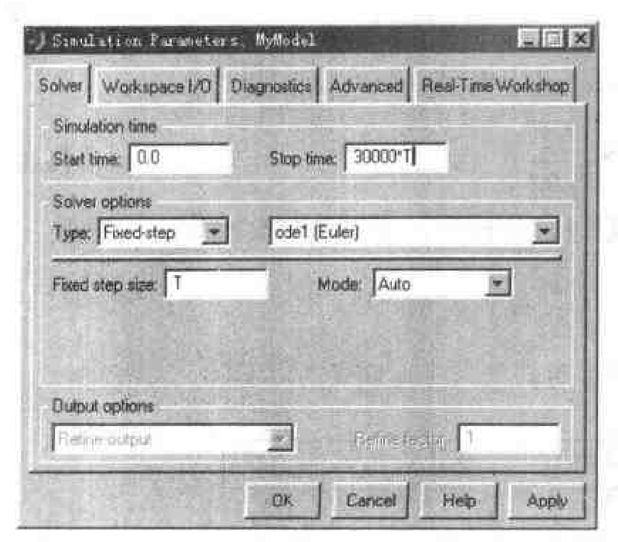
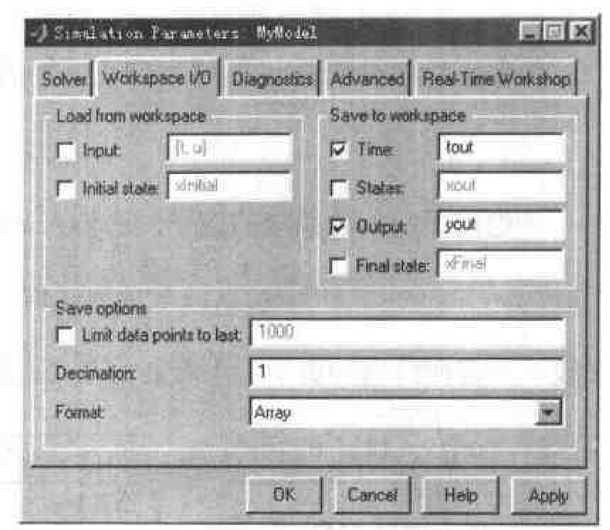


图 4-58 系统仿真模型

Parameters 菜单项，则将得出如图 4-59 (a) 所示的对话框，从中可以选择定步长的 Euler 仿真算法，并令计算步长为 0.1，且选择仿真终止时间为 30000*T，再选择该对话框中的 Workspace I/O 标签，并取消 1000 个返回点的选项，如图 4-59 (b) 所示，则可以开始仿真过程，输出信号的仿真



(a) 仿真算法选择对话框



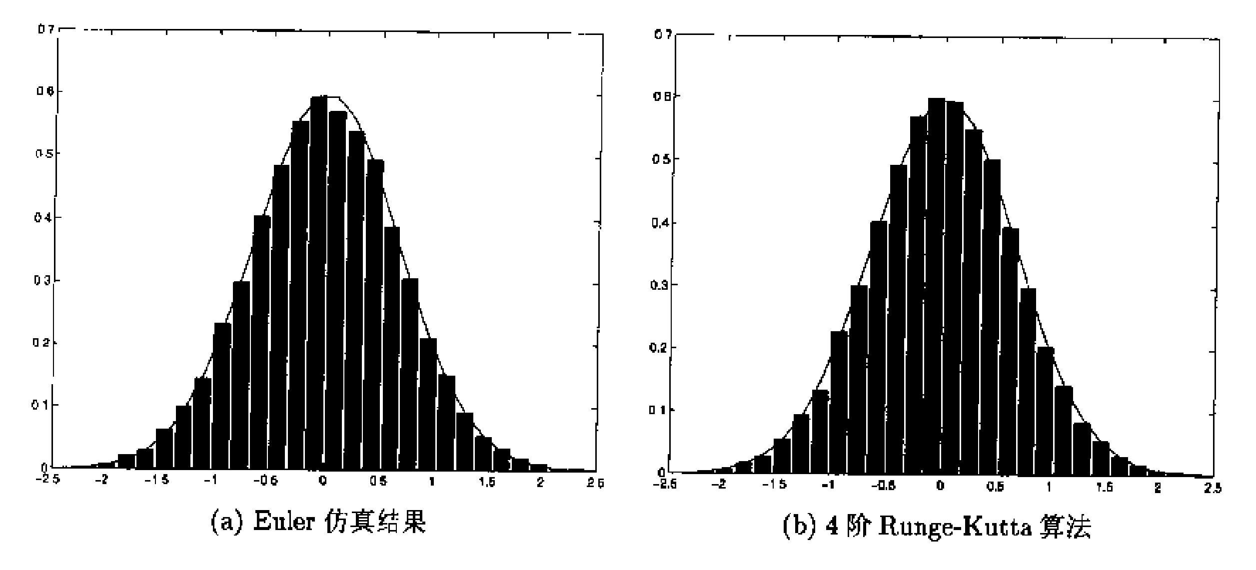
(b) 输出形式设置对话框

图 4-59 仿真参数设置

结果将在 yout 变量中返回 MATLAB 的工作空间，使用下面的命令则可以对输出信号进行概率密

度估算，则结果将如图 4-60 (a) 所示。

```
>> G=tf([1,7,24,24],[1,10,35,50,24]); v=covar(G,1);
   xx=linspace(-2.5,2.5,30); yy=hist(yout,xx);
   yy=yy/(length(yout)*(xx(2)-xx(1)));
   yp=exp(-xx.^2/(2*v))/sqrt(2*pi*v);
   bar(xx,yy), hold on; plot(xx,yp)
```



(a) Euler 仿真结果　　(b) 4 阶 Runge-Kutta 算法

图 4-60　仿真结果比较

选择不同的算法，如选择 4 阶 Runge-Kutta 定步长算法，则使用同样的语句，可以得出如图 4-60(b) 所示的结果，可见，用两种方法得出的结果精度近似。

Simulink 下提供了一个 Band-limited White Noise 输入信号可以生成用于连续系统仿真的白噪声，可以用该模块作输入信号进行仿真，得出的框图如图 4-61 所示。

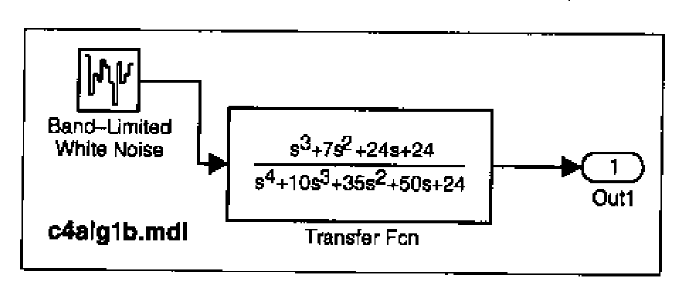
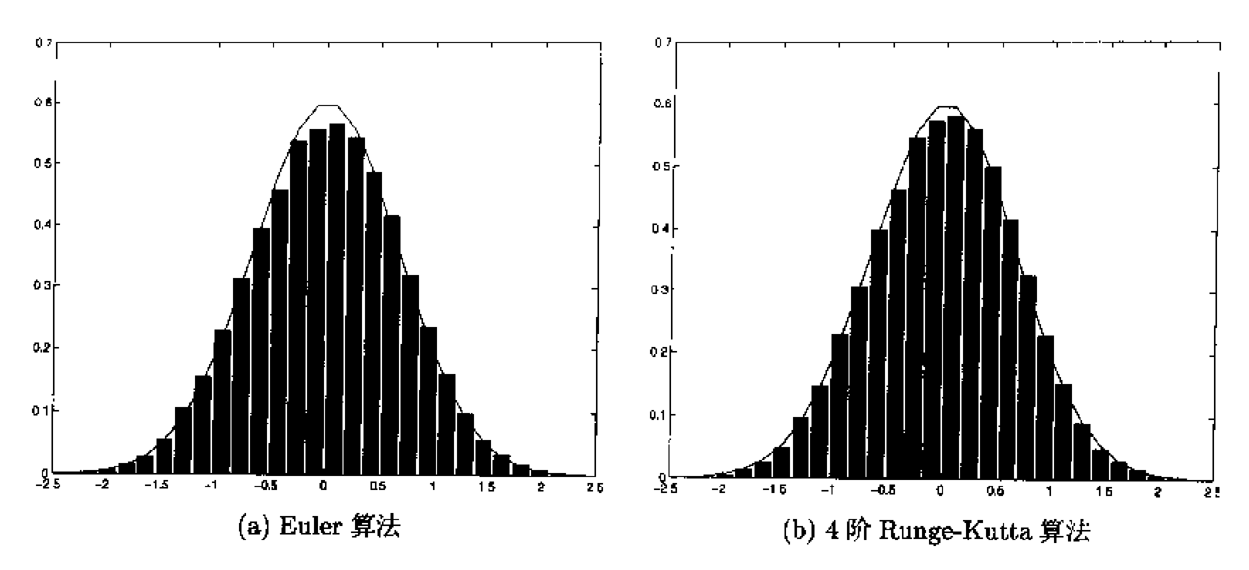


图 4-61　系统仿真模型

将该模块的步长设置为 T，开始仿真过程，再调用相应的命令，则可以得出系统的输出信号的概率密度估算结果如图 4-62 (a) 所示。可以看出，直接采用该模块得出的结果并不是很理想的。将算法算法改为 4 阶 Runge-Kutta 法，则得出的仿真结果如图 4-62 (b) 所示，可以看出，在该算法下仿真结果大有改进。

从前面的例子可见，白噪声信号可以由式 (4.32) 中给出的公式来近似，所以可以将这样的近似算法轻易地推广到非线性系统的仿真中。对一般的随机信号 $N(\mu,\sigma^2)$ 来说，



(a) Euler 算法　　(b) 4 阶 Runge-Kutta 算法

图 4-62　使用 Band-limited White Noise 输入信号的仿真效果

可以按照图 4-63 的格式来构造输入信号。

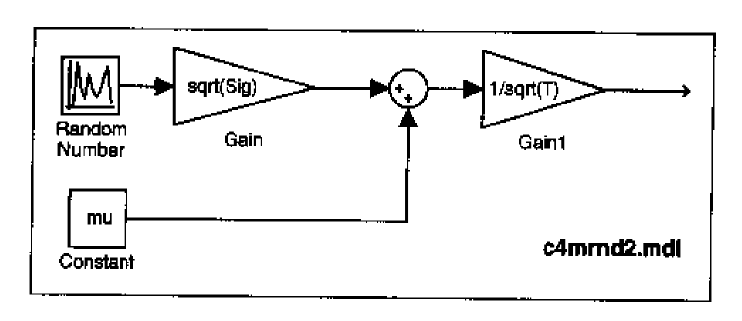


图 4-63　一般随机信号的 Simulink 近似

4.5.3　仿真结果的统计分析

假设线性系统的模型由传递函数 $G(s)$ 给出，则输出信号的自相关函数 $c_{yy}(\tau)$ 可以由双边 Laplace 逆变换公式求出：

$$c_{yy}(\tau)=\frac{S}{2\pi \mathrm{j}}\int_{-\mathrm{j}\infty}^{\mathrm{j}\infty}G(s)G(-s)\mathrm{e}^{s\tau}\mathrm{d}s \tag{4.34}$$

其中 S 为输入白噪声信号的功率谱密度。由谱分解定理，可以将 $G(s)G(-s)$ 分解为：

$$SG(s)G(-s)=\frac{B(s)}{A(s)}+\frac{B(-s)}{A(-s)} \tag{4.35}$$

其中，$A(s)$ 是 $G(s)$ 的分母将多项式零点全部移动到 s-左半平面后的结果，在数学上可以记 $A(s)=D(s)_-$，$D(s)$ 为 $G(s)$ 的分母多项式。这样可以列写出如下方程：

$$B(s)A(-s)+B(-s)A(s)=SN(s)N(-s) \tag{4.36}$$

其中 $B(s)=N(s)_-$, $N(s)$ 为 $G(s)$ 的分子多项式。假设多项式 $A(s)$ 和 $B(s)$ 可以写成

$$A(s)=\sum_{i=0}^{m}\alpha_i s^i,\quad 且\ B(s)=\sum_{i=0}^{m-1}\beta_i s^i \tag{4.37}$$

可以得出

$$\begin{cases} A(s)B(-s)=\displaystyle\sum_{j=0}^{2m-1}\gamma_j s^j,\ 其中,\ \gamma_j=\sum_{k+l=j}(-1)^l\alpha_k\beta_l \\ A(-s)B(s)=\displaystyle\sum_{j=0}^{2m-1}\delta_j s^j,\ 其中\ \delta_j=\sum_{k+l=j}(-1)^k\alpha_k\beta_l \end{cases} \tag{4.38}$$

这样可以得出

$$\begin{aligned} A(s)B(-s)+A(-s)B(s) &= \sum_{k+l=j}(-1)^l\left[1+(-1)^{k-l}\right]\alpha_k\beta_l \\ &= \begin{cases} 2\displaystyle\sum_{k+l=j}(-1)^l\alpha_k\beta_l, & j\ 为偶数 \\ 0, & j\ 为奇数 \end{cases} \end{aligned} \tag{4.39}$$

记 $\mathcal{S}N(s)N(-s)=f_{m-1}s^{2m-2}+f_{m-2}s^{2m-4}+\cdots+f_1s^2+f_0$，则可以由下面的线性代数方程求解未知数 β_i

$$\begin{bmatrix} (-1)^{m-1}\alpha_{m-1} & (-1)^{m-2}\alpha_m & 0 & \cdots & 0 \\ (-1)^{m-1}\alpha_{m-3} & (-1)^{m-2}\alpha_{m-2} & (-1)^{m-1}\alpha_{m-1} & \cdots & 0 \\ (-1)^{m-1}\alpha_{m-5} & (-1)^{m-2}\alpha_{m-4} & (-1)^{m-1}\alpha_{m-3} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \alpha_0 \end{bmatrix} \begin{bmatrix} \beta_{m-1} \\ \beta_{m-2} \\ \beta_{m-3} \\ \vdots \\ \beta_0 \end{bmatrix} = \frac{\mathcal{S}}{2} \begin{bmatrix} f_{m-1} \\ f_{m-2} \\ f_{m-3} \\ \vdots \\ f_0 \end{bmatrix} \tag{4.40}$$

这样可以得出输出信号的自相关函数为

$$c_{yy}(\tau)=\begin{cases} \mathcal{L}^{-1}\left[\dfrac{B(s)}{A(s)}\right], & \tau>0 \\ c_{yy}(-\tau), & \tau\leqslant 0 \end{cases} \tag{4.41}$$

或认为 $c_{yy}(\tau)$ 为 $B(s)/A(s)$ 的脉冲响应。

根据上述的算法，可以方便地编写出计算谱分解的 MATLAB 函数

```
function [B,A] =spec_fac(num,den)
m=length(den)-1;
NN=conv(num,(-1).^[length(num)-1:-1:0].*num);
X=NN(1:2:end)';  X=0.5*[zeros(m-length(X),1); X];
p=roots(den); ii=find(p>0); p(ii)=-p(ii); A=poly(p);
k=0;
```

```
   if m>1
      Xx=[(-1)^(m-1)*A(2) (-1)^m*A(1) zeros(1,m-2)];
   else
      Xx=[A(1)];
   end
   V0=Xx;
   for i=2:m
      V0=[0 0 V0(1:m-2)]; k=k+2;
      if k<m+1,
         V0(2)=(-1)^m*A(k+1);
         if k<m, V0(1)=(-1)^(m-1)*A(k+2); end
      end
      Xx=[Xx; V0];
   end
   B=[inv(Xx)*X]';
```

利用该函数，由给定的传递函数 num 和 den 就能求出满足谱分解的 B 和 A 向量了。

对非线性系统来说，求取其自相关函数就没有这么容易了，可以先对要研究的系统进行仿真，再用第 3 章中介绍的相关分析函数来分析仿真结果，得出自相关函数和感兴趣信号的互相关函数[50]。

〖例 4.13〗 再考虑下面的模型：

$$G(s)=\frac{s^3+7s^2+24s+24}{s^4+10s^3+35s^2+50s+24}$$

假设输入信号的方差为 3，则可以由下面的语句得出输出的自相关函数

```
>> num=[1 7 24 24]; den=[1 10 35 50 24]; G=tf(num,den);
   [B,A]=spec_fac(3*num,den); G1=tf(B,A);
   [y0,t]=impulse(G1,6); % 求取系统的脉冲响应数据
   t=[-t(end:-1:2); t]; y0=[y0(end:-1:2); y0]/max(y0); % 镜像延拓并归一化
```

用下面语句可以求出该信号自相关函数的仿真结果：

```
>> G=tf([1,7,24,24],[1,10,35,50,24]); T=0.02;
   [F,G0,D,C]=sc2d(G,3,T);
   n_point=30000; r=randn(n_point+4,1); r=r-mean(r);
   y=zeros(n_point,1); x=zeros(4,1); d0=0;
   for i=1:n_point
      x=F*x+G0*d0+D*r(i:i+3); y(i)=C*x;
   end % 以上语句可以得出仿真结果
   [Cyy,f]=crosscorr(y,y,300); % 由互相关函数求自相关函数，自动延拓
   f=f*T; plot(f,Cyy,t,y0,':') % 绘制曲线，理论值用虚线表示
```

这样就可以绘制出输出信号的自相关函数曲线，如图 4-64 所示。

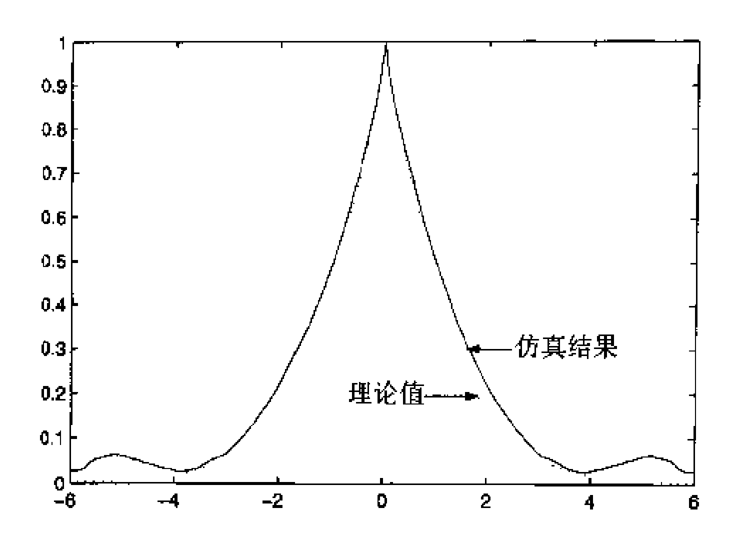


图 4-64 输出信号的自相关函数

线性系统输出信号的功率谱密度是频率 ω 的函数，可以由下式求出

$$G_p(\omega) = G(\mathrm{j}\omega)G(-\mathrm{j}\omega)P \tag{4.42}$$

其中 P 为输入信号的功率谱密度。如果仿真数据已经获得，则也可以用第 **3.6** 节介绍的 **psd_estm()** 函数估算出的功率谱密度。

〖例 4.14〗 再考虑前面得出的仿真结果，可以用下面语句估计出输出信号的功率谱密度及系统功率谱密度理论值，绘制出二者的比较，如图 4-65 所示。可以发现，两种方法得出的结果几乎一致，由此可以看出使用的仿真算法的可靠性。

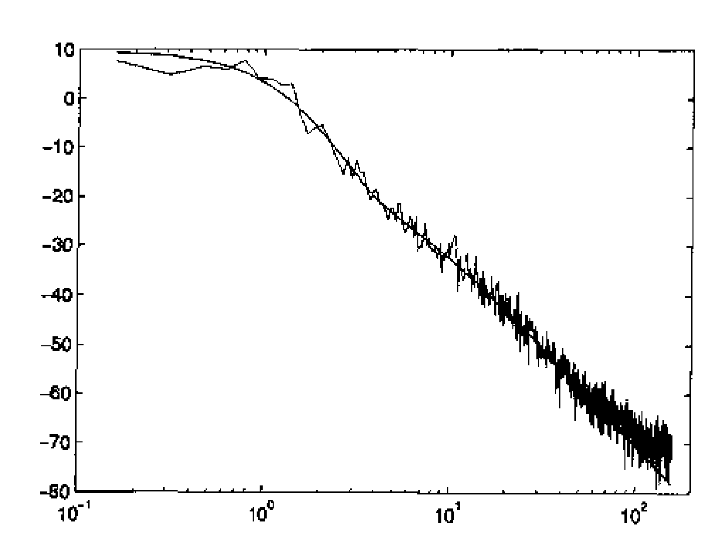


图 4-65 输出信号的功率谱密度比较

```
>> [Pxx,f]=psd_estm(y,2028,T);
   num=G.num{1}.*(-1).^[length(G.num{1})-1:-1:0];
   den=G.den{1}.*(-1).^[length(G.den{1})-1:-1:0];
```

```
GG=3*G*tf(num,den); % 计算 GG(s)=P G(s)G(-s)
[mag,p]=bode(GG,f); % 计算 GG(jw)
semilogx(f,20*log10(mag(:)),f,20*log10(Pxx))
```

4.6 分形系统的仿真

1967 年法国数学家 Mandelbrot 提出了“英国的海岸线有多长”这样的命题，其答案是出乎意料的，“它取决于你的尺子”。海岸线的测量确实和尺子的单位有关，如果用 1 公里为单位，则近似长度小于 1 公里的曲折和迂回就都忽略掉了，如果你减小尺子的单位，则会发现测出的长度在增加。Mandelbrot 发现，当测量单位变得无穷小时，所得到的总长度是无穷大。另外他还发现了海岸线的“自相似性”。在不同比例尺下的地图中，可以发现海岸线的形状大体相同，其曲折、复杂程度是相似的，要研究这样的问题引入分数维数也许是必要的，所以他开创了分形几何的体系，来研究分形 (fractal) 问题。

以前研究的数学问题都是整数维的，更确切的说，是一维、二维和三维的，假设在整数维数下有一个图形，将原图的线度缩小 r 倍，且变换后的图形由 k 个相似的小图形组成，则 r, k 和维数 d 之间存在关系式为 $r^d = k$，从而得出新的维数为：

$$d = \ln k/\ln r \tag{4.43}$$

该值为分数，所以这样构造的图形为分数维图形。

这里将不去探讨分形几何的理论，只通过例子介绍一些有代表性的经典数学模型，如一般分形树的构造、Julia 图和 Mandelbrot 图形等，这里介绍的求解方法主要依赖 MATLAB 编程及其图形可视化技术。可以看出，分形系统的仿真可能创造出各种各样的美妙计算机图形。

4.6.1 分形树的仿真与绘图

分形的特色在于其自相似性，所谓自相似性，是指在图形中的小图形是比较大的图形的按比例缩小后的结果，而更小的图形是这些小图形按比例再缩小后的结果，依此类推。所以，可以先构造一个图形，然后根据它生成各级子图形。

【例 4.15】 按照上面的概念，有人推导出了一个分形树的数学模型：任意选定一个二维平面上的初始点坐标 (x_0,y_0)，假设可以生成一个在 $[0,1]$ 区间上均匀分布的随机数 γ_i，那么根据其取值的大小，可以按下面的公式生成一个新的坐标点 (x_1,y_1):

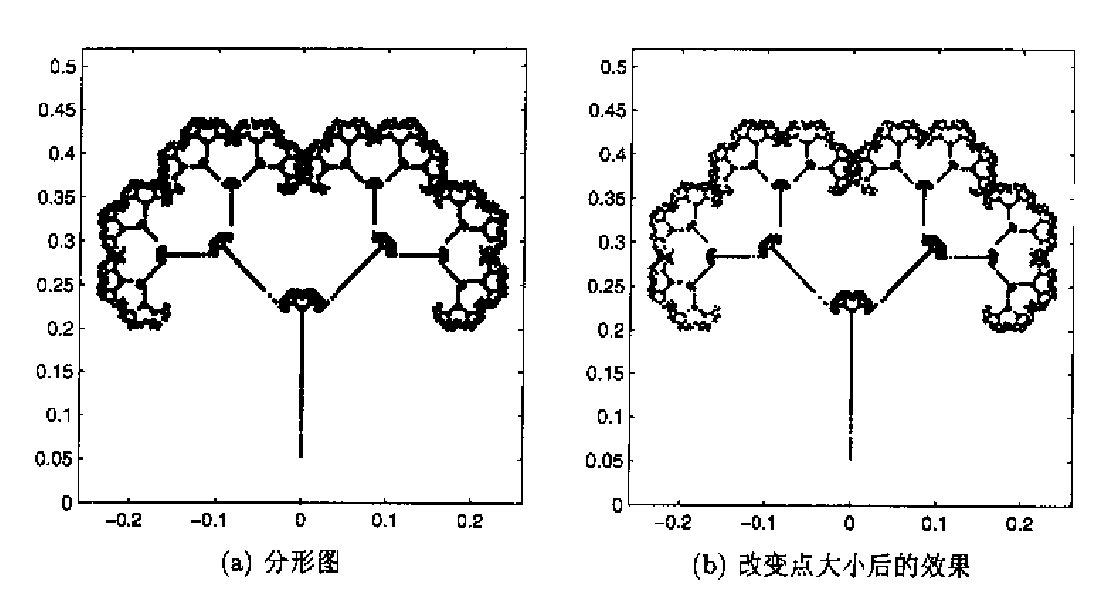
$$(x_1,y_1) \Leftarrow \begin{cases} x_1=0,\ y_1=y_0/2, & \gamma_i<0.05\text{ 时} \\ x_1=0.42(x_0-y_0),\ y_1=0.2+0.42(x_0+y_0), & 0.05\leq\gamma_i<0.45\text{ 时} \\ x_1=0.42(x_0+y_0), y_1=0.2-0.42(x_0-y_0), & 0.45\leq\gamma_i<0.85\text{ 时} \\ x_1=0.1x_0,\ y_1=0.2+0.1y_0, & \text{其他情况} \end{cases}$$

令新生成的点 (x_1, y_1) 为初始点 (x_0, y_0)，可以再生成一个新的点，可以多次重复这样的过程，这样就能生成一族坐标点．假设想根据这样的方式产生 N 个点，则可以由下面的 MATLAB 语句编写出一个函数。

```
function [x,y]=frac_tree(x0,y0,v)
N=length(v);
x=[x0; zeros(N-1,1)]; y=[y0; zeros(N-1,1)];
for i=2:N
   vv=v(i);
   if vv<0.05, y(i)=0.5*y(i-1);
   elseif vv<0.45,
      x(i)=0.42*(x(i-1)-y(i-1)); y(i)=0.2+0.42*(x(i-1)+y(i-1));
   elseif vv<0.85,
      x(i)=0.42*(x(i-1)+y(i-1)); y(i)=0.2-0.42*(x(i-1)-y(i-1));
   else,
      x(i)=0.1*x(i-1); y(i)=0.1*y(i-1)+0.2;
   end
end
```

调用此函数，可以由下面的 MATLAB 命令生成 10,000 个这样的点，并将这些点在 MATLAB 图形窗口中用点的形式表示出来，如图 4-66(a) 所示。

```
>> v=rand(10000,1); [x,y]=frac_tree(0,0,v);
   h=plot(x,y,'.'); axis('square',[-0.26,0.26,0,0.52])
```



(a) 分形图　　(b) 改变点大小后的效果

图 4-66　分形树的 MATLAB 表示

从得出的图形可以看出，该图形中的"树枝"的形状与结构和"树干"完全一致，而再小些的"树枝"的结构和形状也和树干完全一致，这样的树枝结构可以一级一级进行下去，这就说明

了所谓的自相似性。

注意，默认选择的点过大，难以较准确地反映这里的树状图。也可以用 MATLAB 图形窗口的编辑界面改变点的大小，也能由下面的 MATLAB 命令做同样的设置：

```
>> set(h,'MarkerSize',4)
```

这时将其图形点的 MarkerSize 属性的值设为 4，则可以得出如图 4-66(b) 所示的结果。

〖例 4.16〗 在一般分形树的计算中，如果要求的计算点数很大，则纯粹使用 MATLAB 编程会显得执行的速度较慢，所以可以考虑采用 Mex 函数的格式编写计算程序。考虑例 4.15 中给出的问题，更夸张地，若需要计算 1,000,000 个点，则可以给出如下的 MATLAB 命令。

```
>> N=1000000; v=rand(N,1);
   tic, [x,y]=frac_tree(0,0,v); toc
elapsed_time =
   28.0640
```

可以看出，求解这样的问题可能需要花费很长的时间，从而使 MATLAB 在解决这个问题上效率不高。如果依照 Mex 的规则设计一个 C 程序 frac_tree1.c，其源码如下：

```
#include "matrix.h"
void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
                 const mxArray *prhs[])
{
   double *xp, *yp, *vp, *x0p, *y0p, *v0p, x0, y0;
   double vv;
   long i, N;
   x0p=mxGetPr(prhs[0]); y0p=mxGetPr(prhs[1]);
   v0p=mxGetPr(prhs[2]); N=mxGetM(prhs[2]);
   plhs[0]=mxCreateDoubleMatrix(N,1, mxREAL);
   plhs[1]=mxCreateDoubleMatrix(N,1, mxREAL);
   xp=mxGetPr(plhs[0]); yp=mxGetPr(plhs[1]);
   xp[0]=x0p[0]; yp[0]=y0p[0];
   for (i=1; i<N; i++){
      vv=v0p[i];
      if (vv<0.05)
         {yp[i]=0.5*yp[i-1]; xp[i]=0;}
      else if (vv<0.45) {
         xp[i]=0.42*(xp[i-1]-yp[i-1]); yp[i]=0.2+0.42*(xp[i-1]+yp[i-1]);
      }
      else if (vv<0.85) {
         xp[i]=0.42*(xp[i-1]+yp[i-1]); yp[i]=0.2-0.42*(xp[i-1]-yp[i-1]);
      }
      else {xp[i]=0.1*xp[i-1]; yp[i]=0.1*yp[i-1]+0.2;}
```

```
    }
}
```

编译此程序，则可以生成一个 DLL 文件 frac_tree1.dll，这时再对同样的问题调用此函数则可以得出如下结果

```
>> N=1000000; v=rand(N,1);
   tic, [x1,y1]=frac_tree1(0,0,v); toc
elapsed_time =
   0.3010
```

可以看出，同样的问题若使用 Mex 技术，会使得效率大大地提高，对此例来说可以提高近 100 倍的速度。所以在求解运算量大的、尤其是含有循环的问题时，可以采用 Mex 技术来处理。

4.6.2 Julia 图的仿真与绘制

Julia 图形来源于复数的映射，假设有一个映射关系 $z_{n+1}=f(z_n)$，z_n 的值取遍某一复数区域内所有的网格点，则对映射点 z_{n+1} 定义一种测度，根据这样的测度，配以适当的颜色，最终绘制出来相应的图形，该复数映射表达式可以看成是一种系统，这样的系统又称为复动力系统。

〖例 4.17〗 选定一个复数 c，在对 $(x_{\min},y_{\min})$ 到 $(x_{\max},y_{\max})$ 平面区域内每个点 $z_0=x_0+\mathrm{j}y_0$ 做如下映射 $z_{n+1}=z_n^2+c$ 之后，如果变换的点仍为有界的，则再继续上述的映射。进行若干次映射后，可以得出一个新的复数，把它进行变换后赋给 $z(x_0,y_0)$。

首先考虑一种测度，当变换出的点坐标的模大于 a，则以该点处的迭代次数为其测度，这样就可以写出：

```
function W=julia1(X,Y,c,n_iter,a)
Z=X+i*Y; W=zeros(size(X));
for k=1:n_iter,
   Z=Z.^2+c;
   i0=find(abs(Z)>a); W(i0)=k; Z(i0)=NaN;
end
i0=find(W==0); W(i0)=NaN;
```

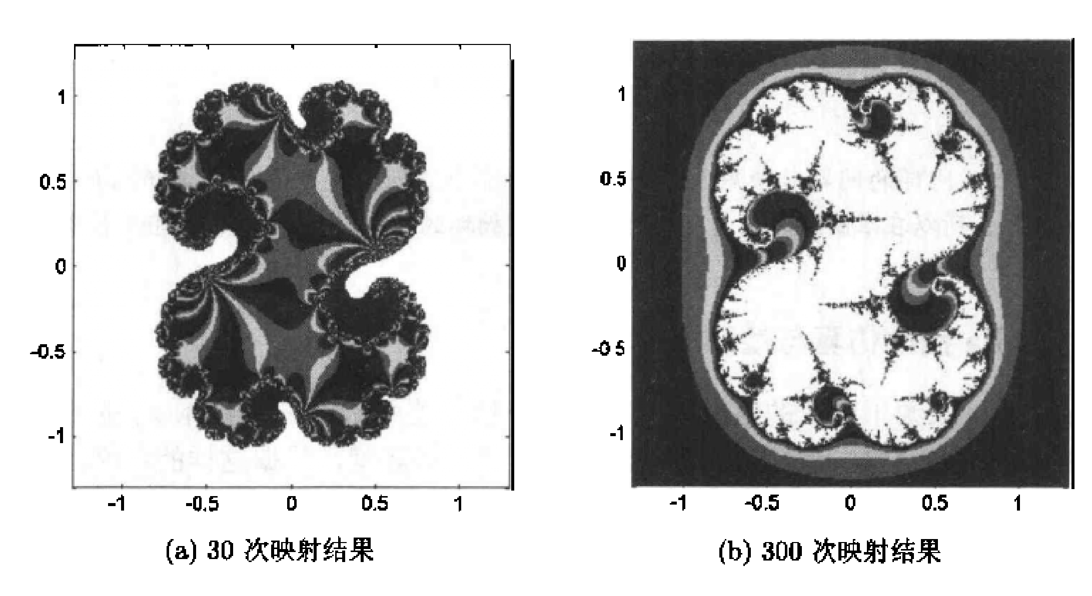
其中，X 和 Y 为网格点的坐标矩阵，n_iter 为最高迭代次数。在实际处理时，如果某个点的模大于 a，则将 W 相应的值设置为迭代次数，同时将对应的 Z 坐标设置成 NaN，使之以后不再参加运算。循环结束后，如果该点坐标的模仍然小于 a，则设其测度值为 NaN。

选择 $a=2$，且 $c=0.27334+\mathrm{j}0.00742$，先选择最大迭代次数为 30，可以用下面的语句得出获得 Julia 图形所需要的值

```
>> x=linspace(-1.3,1.3,300); y=x; [X,Y]=meshgrid(x,y);
   n_iter=30; c=0.27334+0.00742i;
   tic, W=julia1(X,Y,c,n_iter,2); toc
elapsed_time =
```

```
    2.0900
```

迭代 30 次所需的时间大约为 2 秒。获得了数据后，可以使用 MATLAB 提供的伪色彩图形绘制函数 pcolor() 表现出相应的数据，如图 4-67 (a) 所示。这里使用了 prism() 颜色表来表现图形的颜色，当然还可以使用其他的颜色表来显示得出的图形。



(a) 30 次映射结果　　(b) 300 次映射结果

图 4-67　不同映射次数下的 Julia 图

```
>> pcolor(X,Y,W), shading flat; axis('square'); colormap prism(256);
```

增加最大的迭代次数限制为 300，则可以绘制出如图 4-67 (b) 所示的图形，可见二者的差异较大，说明原来最大迭代次数的限制过小。

还可以定义一个测度 $\hat{z}(x_0,y_0) = e^{-|z|}$。很显然，用 MATLAB 处理后，如果某个映射点是无界的，则其值将自动变换成 NaN，从而在图形绘制时被剪切掉。根据这样的算法编写一个 MATLAB 函数 julia()，其清单为：

```
function W=julia(X,Y,c,n_iter)
Z=X+i*Y;
for k=1:n_iter, Z=Z.^2+c; end
W=exp(-abs(Z));
```

其中，X 和 Y 可以是 $(x_{\min},y_{\min})$ 到 $(x_{\max},y_{\max})$ 平面区域内按某种方式分隔的网格，c 为指定的常数 c, n_iter 是允许最大的映射次数，返回的 W 是 $\hat{z}(x_0,y_0)$ 的值构成的矩阵。

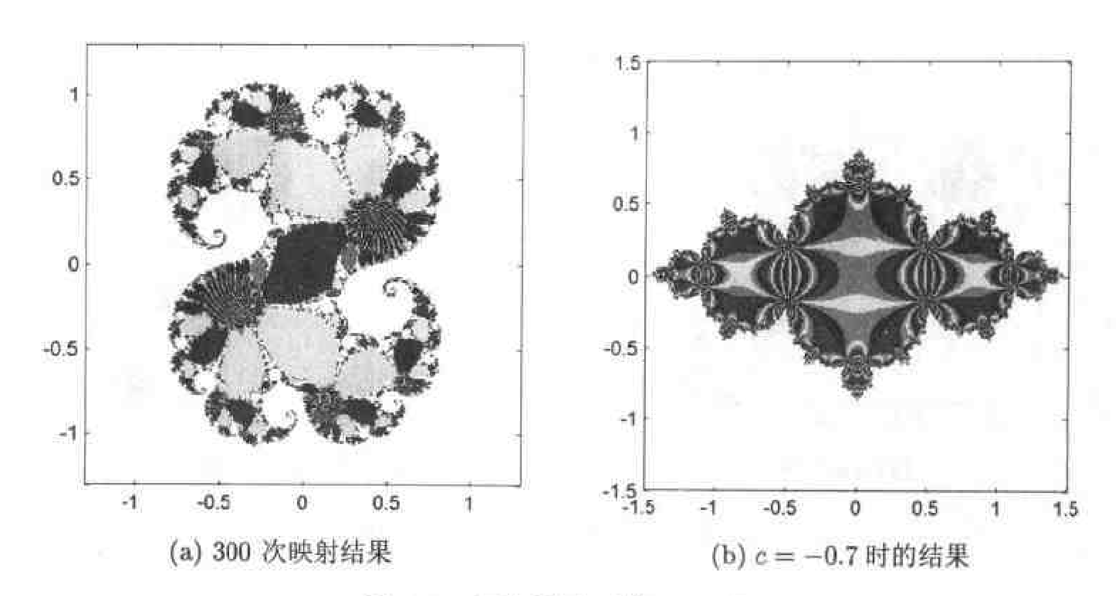
选择 300 次映射，可以给出下面的 MATLAB 语句来计算 Julia 集。

```
>> x=linspace(-1.5,1.5,300); y=x; [X,Y]=meshgrid(x,y);
   n_iter=300; W=julia(X,Y,c,n_iter);
```

这样得出的 Z 矩阵中有很多元素为 NaN。MATLAB 在解决这样问题中的优势就是它能有效地处理计算过程中的 NaN，所以用 MATLAB 语言来计算这样的问题是合适的。

可以调用 pcolor() 函数用伪彩色描述得出的 Z 矩阵。如果将色调设置成 prism()，则最终可以由下面的 MATLAB 语句绘制出 Julia 分形图，如图 4-68(a) 所示。

```
>> pcolor(X,Y,W), shading flat; axis('square'); colormap prism(256);
```



(a) 300 次映射结果　　(b) $c=-0.7$ 时的结果

图 4-68 不同测度下的 Julia 图

选择其他的 c 值，如 $c=-0.7$，则可以得出如图 4-68(b) 所示的 Julia 图。

```
>> c=-0.7; W=julia(X,Y,c,n_iter);
   pcolor(X,Y,W), shading flat; axis('square'); colormap prism(256);
```

考虑另一个映射函数，$z_{n+1}=(2z_n^3+1)/(3z_n^2)$，编写出求解 Julia 集的函数:

```
function [W,Z]=julia2(X,Y,a,n_iter)
Z=X+i*Y;  W=zeros(size(X));
for k=1:n_iter,
   Z=(2*Z.^3+1)./(3*Z.^2);
   i0=find(abs(Z)>a); W(i0)=k; Z(i0)=NaN;
end
i0=find(W==0); W(i0)=NaN;
```

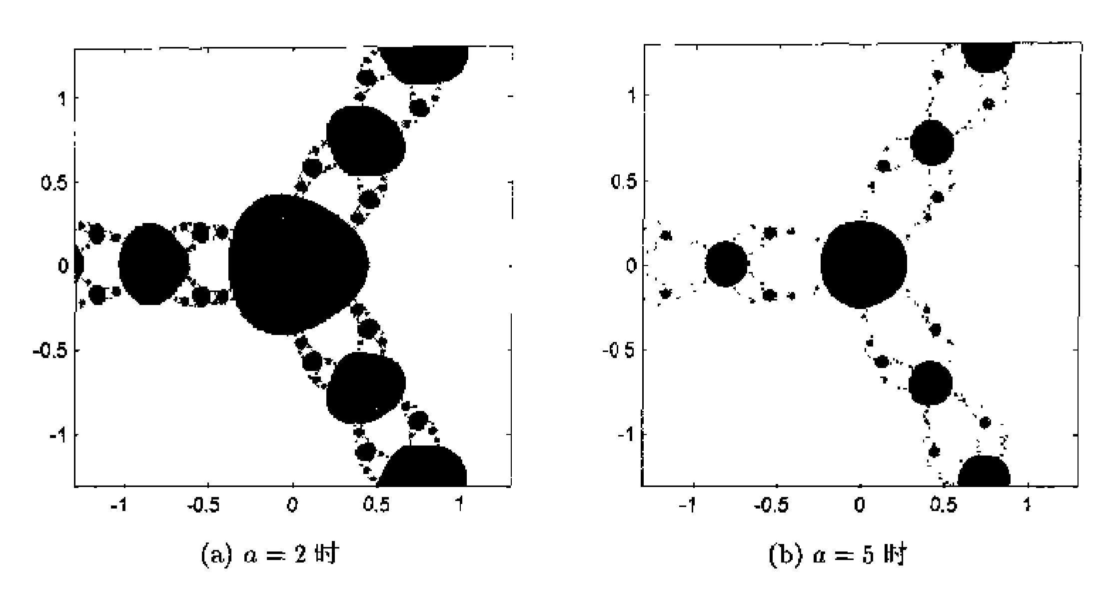
这样可以用下面的语句得出 $a=2$ 时这一映射函数下的 Julia 图，如图 4-69 (a) 所示。

```
>> W=julia(X,Y,2,n_iter);
   pcolor(X,Y,W), shading flat; axis('square'); colormap prism(256);
```

若取 $a=5$，则可以得出如图 4-69 (b) 所示的 Julia 图。

4.6.3 Mandelbrot 图的仿真与绘制

Mandelbrot 图形很可能是分数维图形世界中已被开发的最著名的图形，和 Julia 图类似，在 $z_{n+1}=z_n^2+c$ 映射中，如果 z_0 遍取区域内 $(x_{\min},y_{\min})$ 到 $(x_{\max},y_{\max})$ 所有的点，则可以得出各个点处的测度，从而绘制出 Mandelbrot 图。选定不同的 z_0 值变化区域，则可以绘制出各种各样美妙的 Mandelbrot 图形。



(a) $a=2$ 时　　(b) $a=5$ 时

图 4-69　不同 a 值下的 Julia 图

〖例 4.18〗 根据上面介绍的算法，可以立即写出 Mandelbrot 集的求取函数：

```
function W=mandelbrot(X,Y,c,a,n_iter)
C=X+i*Y; W=zeros(size(X)); Z=c;
for k=1:n_iter,
   Z=Z.^2+C;
   i0=find(abs(Z)>a); W(i0)=k; Z(i0)=NaN;
end
i0=find(W==0); W(i0)=NaN;
```

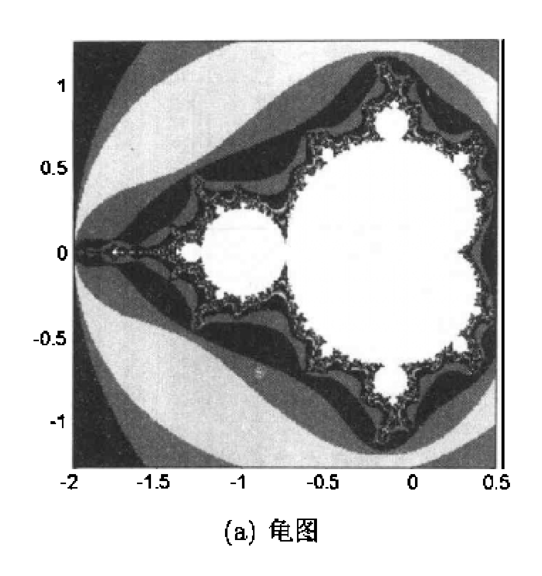
其中，c 为 c 的值，a 为收敛阈值，这样就可以由下面的命令绘制出相应的 Mandelbrot 图，如图 4-70 (a) 所示，该图即著名的"龟图"[48]。

```
>> x=linspace(-2,0.5,300); y=linspace(-1.25,1.25,300);
   [X,Y]=meshgrid(x,y); n_iter=200; c=0; a=2;
   W=mandelbrot(X,Y,c,a,n_iter);
   pcolor(X,Y,W), shading flat; axis('square'); colormap prism(256);
```

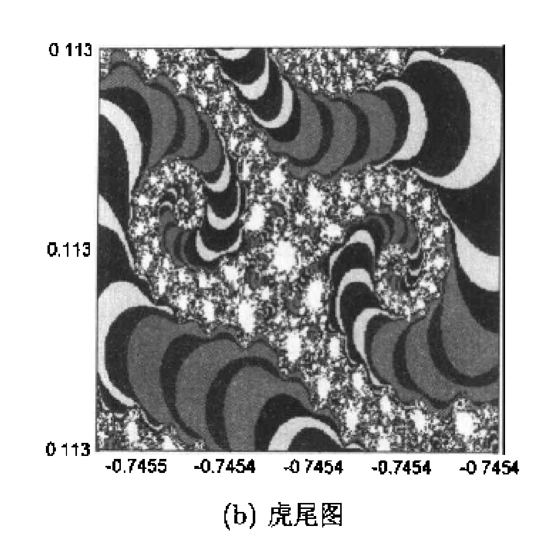
选择其中一个小的区域，再调用 mandelbrot() 函数

```
>> x=linspace(-0.74547,-0.74538,300); y=linspace(0.11298,0.11304,300);
   [X,Y]=meshgrid(x,y); n_iter=200; c=0; a=2;
   W=mandelbrot(X,Y,c,a,n_iter);
   pcolor(X,Y,W), shading flat; axis('square'); colormap prism(256);
```

则可以绘制出"放大"的 Mandelbrot 图，如图 4-70 (b) 所示，该图又称为"虎尾图"。可以看出，利用这里给出的函数就能绘制出任意区域内的 Mandelbrot 图。



(a) 龟图



(b) 虎尾图

图 4-70　不同 Mandelbrot 图

4.7　习　题

(1) 在标准的 Simulink 模块组中，各个模块组中的模块遵从比较好的分类方法，请仔细观察各个模块组，熟悉其模块构成，以便以后遇到某些需要时能迅速、正确地找出相应的模块，容易地搭建起 Simulink 模型。

(2) 考虑简单的线性微分方程

$$y^{(4)}+3y^{(3)}+3\ddot{y}+4\dot{y}+5y=\mathrm{e}^{-3t}+\mathrm{e}^{-5t}\sin(4t+\pi/3)$$

且方程的初值为 $y(0)=1,\dot{y}(0)=\ddot{y}(0)=1/2,y^{(3)}(0)=0.2$，试用 Simulink 搭建起系统的仿真模型，并绘制出仿真结果曲线。

(3) 考虑上面的模型，假设给定的微分方程变化成时变线性微分方程：

$$y^{(4)}+3ty^{(3)}+3t^2\ddot{y}+4\dot{y}+5y=\mathrm{e}^{-3t}+\mathrm{e}^{-5t}\sin(4t+\pi/3)$$

而方程的初值仍为 $y(0)=1,\dot{y}(0)=\ddot{y}(0)=1/2,y^{(3)}(0)=0.2$，试用 Simulink 搭建起系统的仿真模型，并绘制出仿真结果曲线。

(4) 建立起如图 4-71 所示非线性系统[20] 的 Simulink 框图，并观察在单位阶跃信号输入下系统的输出曲线和误差曲线。

(5) 在例 3.29 中给出了 Lorenz 方程的数值解法，例 3.35 则给出了 Apollo 方程表示的运行轨道问题求解，试使用 Simulink 中提供的模块搭建出这两个例子中的微分方程，并将得出的结果和前面例子相比较。

(6) 例 3.36 中求解方程时，要求用户首先进行预处理，将原来的高阶微分方程组变换成一阶微分方程组，试用 Simulink 建立起相应的模型，观察是否还应该进行相应的预处理。

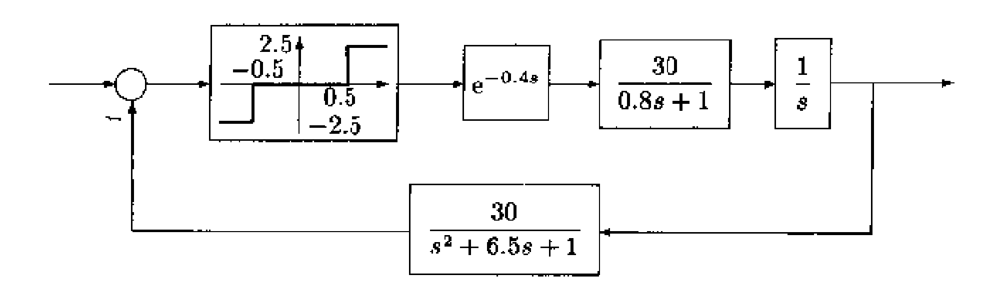


图 4-71　习题 (4) 图

(7) 建立起如图 4-72 所示非线性系统[47] 的 Simulink 框图，并观察在单位阶跃信号输入下系统的输出曲线和误差曲线。另外，本系统中涉及到两个非线性环节的串联，试问这两个非线性环节可以互换吗？试从仿真结果上加以解释。

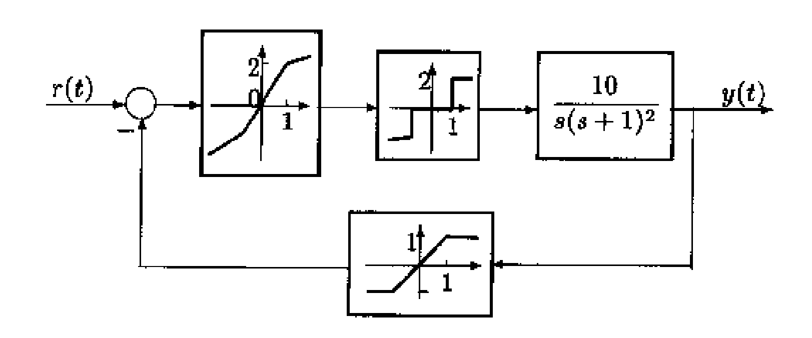


图 4-72　习题 (7) 图

(8) 请用 Simulink 搭建出下面的数字逻辑电路

(a) $Z_1 = A + B\overline{C} + D$,　(b) $Z_2 = AB(C+D) + D + \overline{D}(A+B)(\overline{B}+\overline{C})$

并自己选定信号验证这两个电路是等价的。

(9) 上一章的习题中给出了 Rössol 数学模型，试用 Simulink 搭建该模型，进行仿真分析，并将结果和上章得出的结果相比较。

(10) 假设已知线性系统模型为

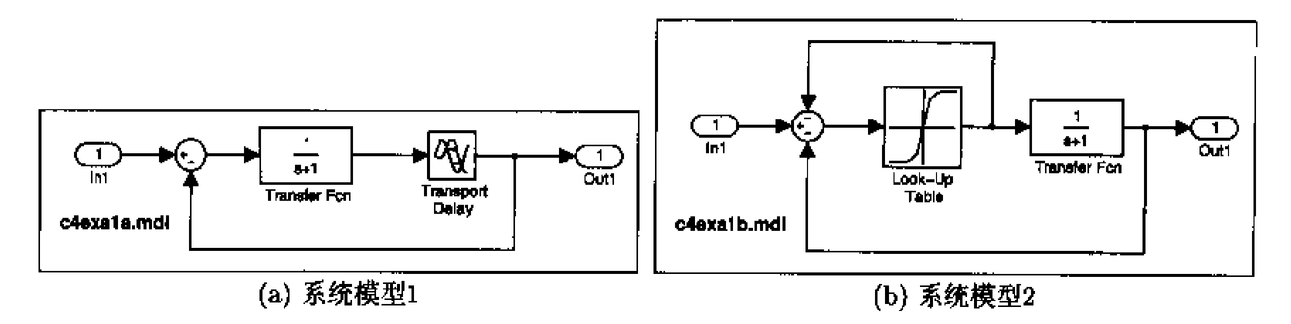
$$G(s) = \frac{s^2+5s+2}{(s+4)^4+4s+4}$$

试输入该系统模型，并求出系统在脉冲输入、阶跃输入和斜坡输入下的解析解，并和仿真曲线相比较，验证得出的结果。

(11) 对例 4.4 中给出的采样系统进行时域和频域分析。

(12) 对例 4.2 中给出的直流电机拖动系统进行开环频域响应曲线绘制，如绘制出系统的 Bode 图，Nyquist 图和 Nichols 图。

(13) 考虑图 4-73 中给出的两个 Simulink 模型，试分析这两个模型是否含有代数环，试进行仿真验证，并分析如果存在代数环是否影响仿真结果。



(a) 系统模型1　　(b) 系统模型2

图 4-73　习题 (13) 图

(14) 对线性传递函数模型

$$G(s)=\frac{6(s+1)}{(s+2)(s+3)(s+4)}$$

在白噪声信号 $\gamma(t)$ 激励下的行为进行仿真，求出其输出函数的概率密度 曲线，并和由仿真结果求出的结果进行比较；另外，对此系统进行自相关函数和 概率谱密度分析，并比较仿真结果和理论结果是否一致。

(15) 从某一个初值点 x_0, y_0 出发，可以按照下面的公式

$$\begin{cases} x_{n+1}=y_n \\ y_{n+1}=1.16\sin[0.9999\sin(4y_n)+\sin x_n]+0.69999x_n \end{cases}$$

其中 $n=1,2,\cdots,N$，N 可以取一个大的数值，将这样得出的一些数据用描点的方法绘制出来，观察其效果。

(16) 试在 $[-1.4,-1.4]-[1.4,1.4]$ 区域下绘制 $c=-0.12+\mathrm{j}0.74$ 时的 Julia 图。如果将映射公式变成 $z_{n+1}=2z_n^3+c$，试绘制出相应的 Julia 图。

(17) 试绘制如下区域的 Mandelbrot 图：

(a) $[-0.95,0.23333]-[-0.88333,0.3]$，(b) $[-0.74745,0.10671]-[0.74611,0.10779]$ (深潭图)

(18) 在分形系统图形绘制中，采用颜色映射函数 `prism()` 来表示色调，其实，`colormap` 函数还可以调用其他的颜色映射函数，试观察在其他色调下这些图形的效果。

第 5 章 Simulink 常用模块介绍与应用技巧

在第 4 章中介绍了简单 Simulink 模型的绘制方法和系统仿真的方法，在第 5.1 节中将首先用简单的例子更进一步地演示模型绘制知识与技巧，包括向量化的模型输入方法、线性模型的输入和仿真、非线性模块及查表模块、开关模块的应用等，并将演示微分代数方程在 Simulink 环境中的搭建方法及求解。第 5.2 节将介绍输出模块的使用，如输出信号的各类示波器输出、工作空间变量输出与表盘与量计输出，并将介绍输出信号的数字信号处理等。第 5.3 节将详细介绍子系统的概念和子系统的封装技术，并介绍自建模型库的方法和技巧，本节还将介绍子系统的控制问题，并通过一个例子来介绍子系统的构造及其在复杂系统建模中的应用。第 5.4 节中将介绍电力系统模块集 及其在一般电路仿真、功率电子系统仿真以及在电机系统仿真中的应用。第 5.5 节将介绍非线性系统设计模块集，并基于其优化涉及思路演示该模块集在不确定系统 PID 控制器设计中的应用。第 5.6 节将通过一个汽车发动机系统的例子进一步地全面介绍复杂系统的 Simulink 模型搭建与剖析等内容。

5.1 常用模块应用技巧

5.1.1 向量化模块举例

在当前版本的 Simulink 中，很多模块不但支持单个的信号输入，还支持向量型的信号输入，亦即将多路的信号通过 Mux 环节合成一路信号，从而直接输入给这类模块。下面将通过几个例子来演示向量化的模块及其应用。

〖例 5.1〗 在例 4.1 中曾给出了 Van der Pol 方程的 Simulink 表示。事实上，如果将其状态方程写成下面的向量形式

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\mu(x_1^2 - 1)x_2 - x_1 \end{bmatrix}$$

则可以用单个积分器模块来完成向量化建模，得出的 Simulink 模型如图 5-1 所示。

注意，在向量化建模中，可以将向量的各个组成信号通过 Demux 模块进行分解，然后可以单独对各路信号进行运算，再将结果用 Mux 模块汇合成单路的信号，传输给向量化的模块进行处理。

在这个系统下只需将积分器模块的初值也设置为列向量的形式，即可对原系统进行仿真，得出的结果将与例 4.1 中得出的完全一致．还可以进一步简化仿真模型，例如 Fcn 模块来生成方程中第 2 个式子的非线性运算，得出如图 5-2 所示的框图，其中，在 Fcn 模块中填写 `-mu*(u[1]*u[1]-1)*u[2]-u[1]`。遗憾的是，该模块并不支持向量化的输入，否则框图实现将更加简单．在该仿真模型运行时，将给出警告信息，“Warning: Output port 1 of block

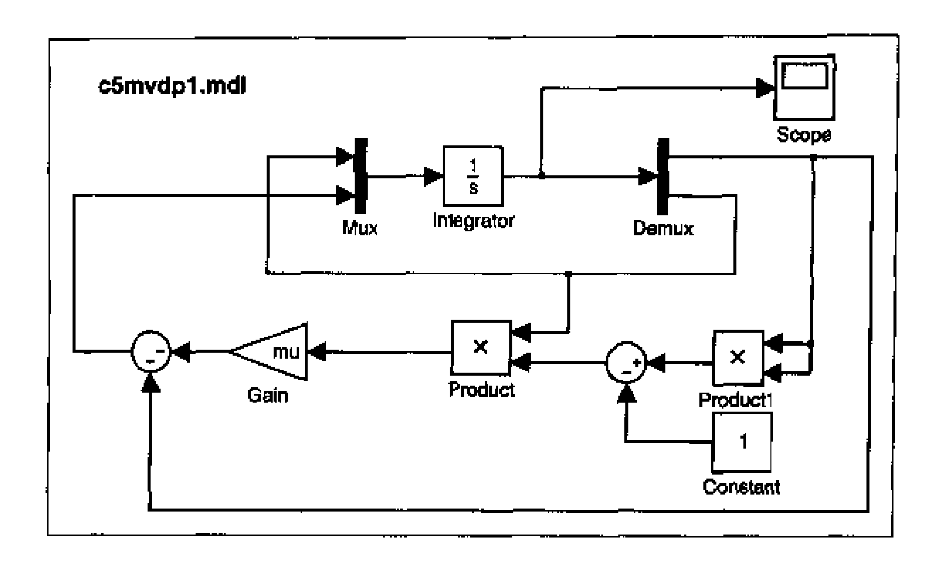


图 5-1 Van der Pol 方程的 Simulink 向量化表示

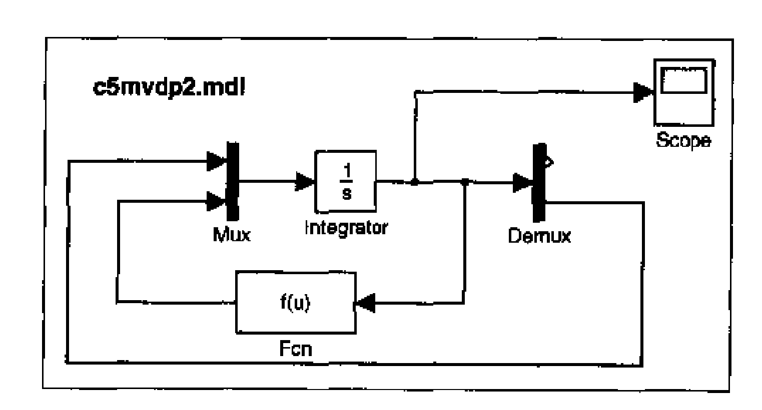


图 5-2 Van der Pol 方程的更简洁向量化表示

'c5mvdp2/Demux' is not connected.",说明该框图中的 Demux 的第 1 端子没有连接,但这将不影响整个过程的正确运行。如果不想得到这样的警告信息,则只需用 Simulation | Simulation parameters 菜单项打开仿真参数对话框,在 Diagnostics 标签中将 Unconnected block outport (未连接的模块输出端口) 的 Action (警告类型) 设置成 None。如果不想做这样的设置,还可以在该端口上接一个信号终结环节 (Terminator),该环节也同样能避免显示这样的警告信息。

〖例 5.2〗 前面介绍过,Simulink 可以容易地建立起线性系统的状态方程模型,但这些模块并不能直接得出系统的内部状态,所以可以用 Simulink 模块搭建起带有状态变量输出的新状态方程模型,如图 5-3 所示。注意,这里所使用的增益模块是矩阵增益 (Matrix Gain),而不能选择一般的标量增益模块。另外,在模型设计时并不存在任何局限性,所以这样建立起来的模型应该适合于任意的连续多变量线性系统。

假设双输入双输出系统的状态方程表示为

$$\dot{x}=\begin{bmatrix}2.25 & -5 & -1.25 & -0.5\\ 2.25 & -4.25 & -1.25 & -0.25\\ 0.25 & -0.5 & -1.25 & -1\\ 1.25 & -1.75 & -0.25 & -0.75\end{bmatrix}x+\begin{bmatrix}4 & 6\\ 2 & 4\\ 2 & 2\\ 0 & 2\end{bmatrix}u,\ \ y=\begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 2 & 0 & 2\end{bmatrix}x$$

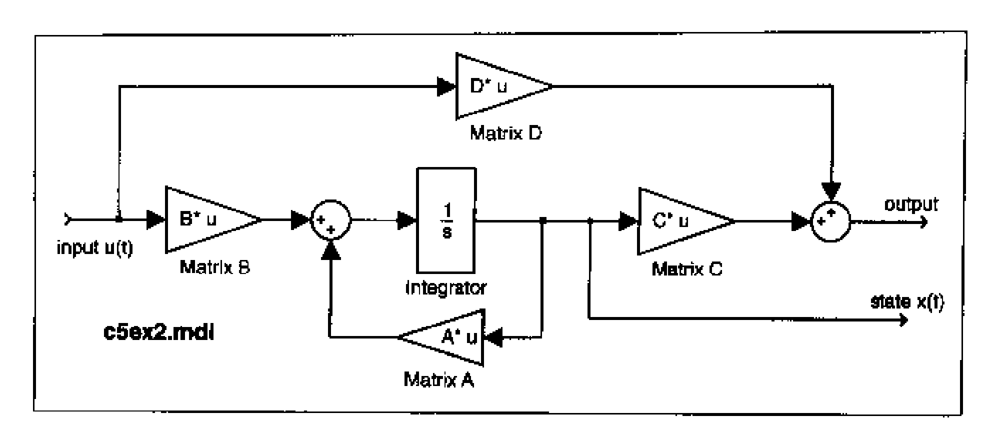


图 5-3 带有状态变量输出的状态方程模型

且输入信号分别为 $\sin t$ 和 $\cos t$，可以通过下面的方式构造出该系统的仿真框图，如图 5-4 所示，

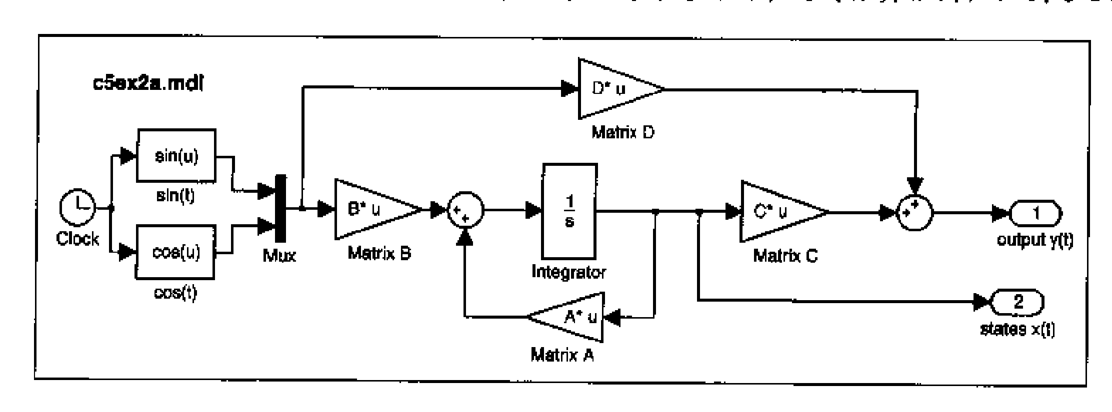


图 5-4 状态方程模型实际仿真框图

当然在仿真前还需输入系统的状态方程模型参数

```
>> A=[2.25, -5, -1.25, -0.5;  2.25, -4.25, -1.25, -0.25;
      0.25, -0.5, -1.25,-1;  1.25, -1.75, -0.25, -0.75];
   B=[4, 6; 2, 4; 2, 2; 0, 2];
   C=[0, 0, 0, 1; 0, 2, 0, 2]; D=zeros(2,2);
```

对整个系统进行仿真，则可以得出 tout 和 yout 两个变量，其中 yout 的前 2 列为系统的输出信号，后 4 列为系统的状态变量，用下面语句则可以得出如图 5-5 所示的曲线。

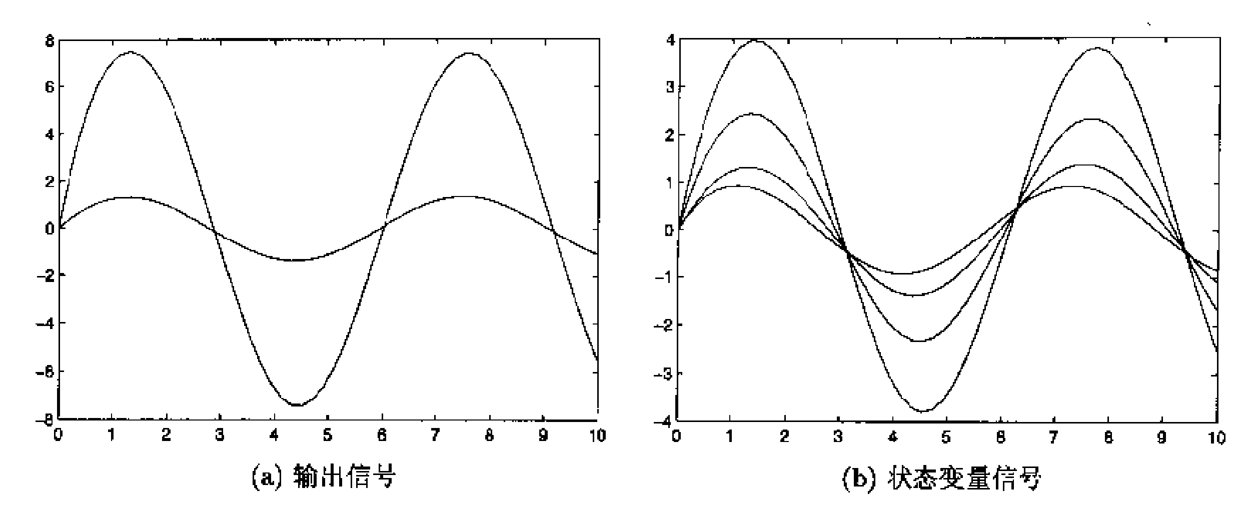
```
>> plot(tout,yout(:,1:2)); % 系统的输出曲线
   figure; plot(tout,yout(:,3:6)) % 系统的状态曲线
```

如果想利用 Simulink 自身的状态方程模块，还同时想获得系统的状态变量，则可以增广 C 和 D 矩阵，使之等于

```
>> C=[C; eye(4)]; D=[D; zeros(4,2)];
```

这样就可以按照图 5-6 所示的方式去构造仿真模型，并将 Demux 的参数设置为 [2,4]，该系统得出的结果将和上面的完全一致。

假设未增广 C 和 D 矩阵，则运行仿真程序时，除了得出错误信息窗口外，还将给出如图 5-7 所示的系统框图，在系统框图中给出了系统向量信号的路数，并用红色标出出错的环节。用户可



(a) 输出信号 (b) 状态变量信号

图 5-5 双输入双输出系统的仿真结果

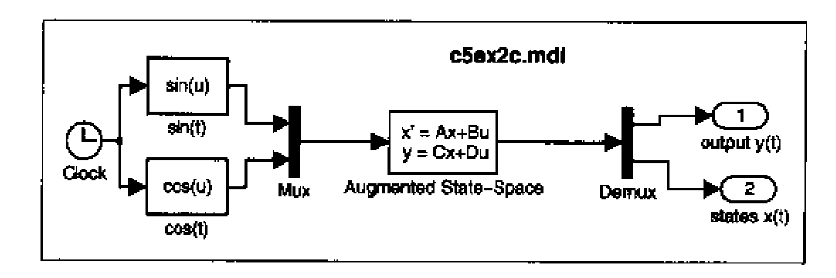


图 5-6 采用状态方程模块的仿真模型

以通过检查出错部分的路数冲突。例如在本例中，Demux 左侧的输入量为 2 路，而右侧第 2 输出就有 4 路，显然造成冲突。Simulink 允许用户通过这样的方式查询错误的原因。

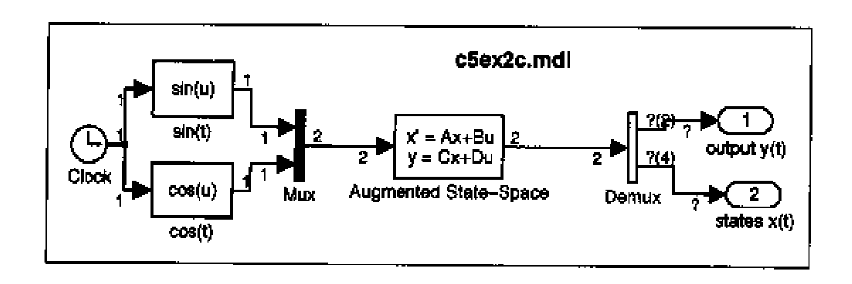
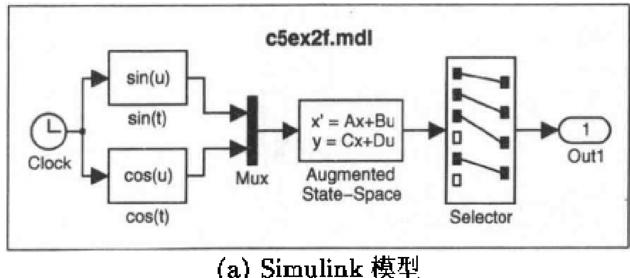
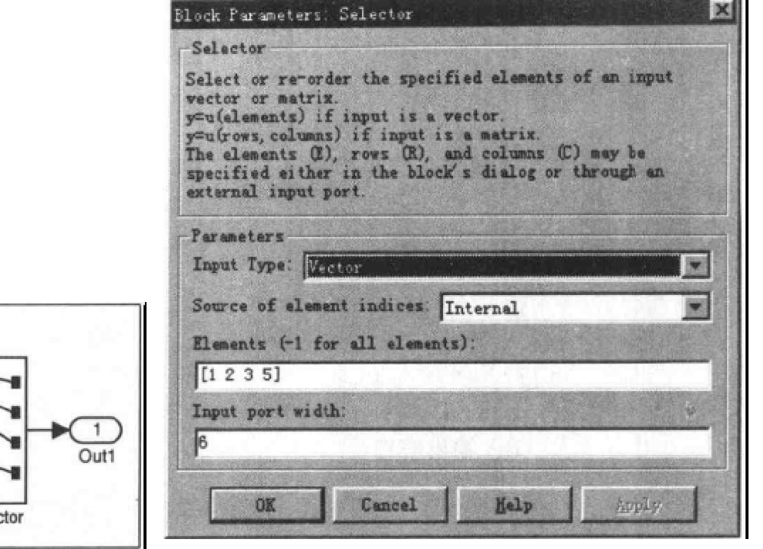


图 5-7 仿真错误信息显示

信号与系统模块组中的选路器 (Selector) 允许用户有选择地输出一些信号，例如，若想显示系统的两路输出与第 1, 3 状态变量，则可以建立起如图 5-8 (a) 所示的仿真框图。双击选择器模块，将得出如图 5-8 (b) 所示的对话框，可以在 Inout port width 栏目中填写 6 (总共输入路数，注意该数值必须和实际输入信号一致，否则将出现错误)，在 Elements 栏目填写输入到输出的对应关系。



(a) Simulink 模型



(b) 选择器参数设置

图 5-8 带有选择器模块的系统

5.1.2 Simulink 模型的信号标识

为了增加 Simulink 模型的可读性，便于了解系统的内部特征，Simulink 环境中提供了若干对模型表示的修饰选项。再看一下如图 4-16 (b) 所示的 Simulation | Format 菜单，在其最后一组菜单下有如下选项：

- Sample time colors (**用不同颜色表示采样信号**) —— 将系统模型中带有不同采样周期的信号用不同的颜色表示出来，以利于系统框图的理解。
- Wide nonscalar lines (**向量化信号线加粗**) —— 在 Simulink 框图中经常出现一些向量化的信号，选择此选项后会自动将这样的信号用粗线表示。
- Signal dimensions (**标明信号路数**) —— 将向量化信号的路数在模块输入和输出处用小字标识。考虑图 5-1 中给出的 Simulink 模型，选择了“加粗”和“信号路数”选项后将得出如图 5-9 所示的图形效果。为方便起见，以后的 Simulink 框图中也将选中这两个选项。
- Port data types (**显示信号的数据类型**) —— 在框图中将显示各路信号的数据类型，如用 double 显示最常用的双精度数据，而 double(2) 表示两路向量化的双精度信号。
- Storage class (**对象类型显示**) —— 将显示其中涉及的对象模块。
- Execution order (**模块排序显示**) —— 在 Simulink 模型进行仿真前，将对每个模块自动进行排序，选择了此选项后，将在每个模块的右上角处显示该模块在仿真框图中的排序序号。仍考虑上述的 Simulink 模型，再选择“数据类型”和“排序”选项后，将得出如图 5-10 所示的显示效果。

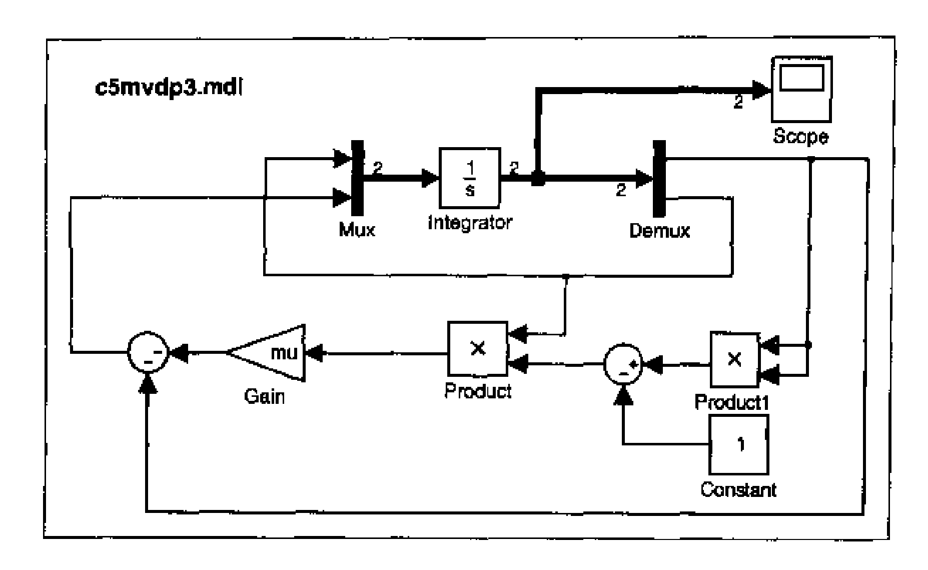


图 5-9 选中了两个修饰选项

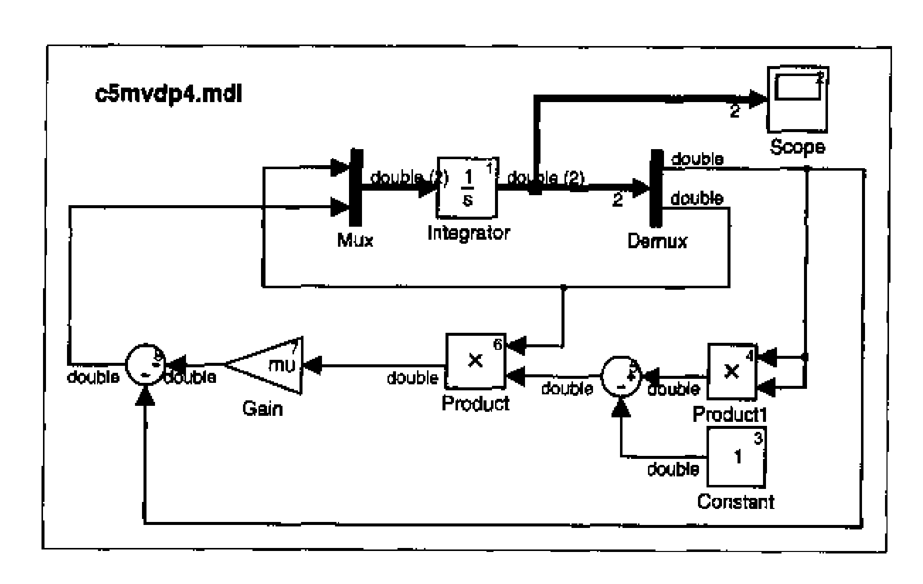


图 5-10 再选中两个修饰选项

5.1.3 线性系统模块

标准的 Simulink 环境中提供了各种各样的线性系统模块，如传递函数模块、状态方程模块与零极点 模块等，此外还有连续系统和离散系统的分别，有时使用起来并非很方便。在 MATLAB 的控制系统工具箱中定义了各种各样的线性时不变 (linear time-invariant，简记为 LTI) 对象，如传递函数对象、状态方程对象和零极点对象，可以用单个变量名来表示整个系统模型。

构造线性系统模型可以分别用下面的语句格式来完成：

```
G=tf(num,den);   G=ss(A,B,C,D);   G=zpk(zer,pol,K)
```

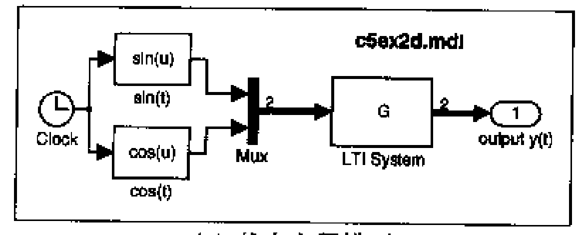
其中，num 和 den 分别为传递函数的分子和分母多项式系数按降幂顺序排列所构成的向量。有了传递函数的分子和分母，则可以用上面的第一个语句建立起传递函数对象。如

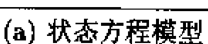
果已知系统的状态方程模型，则可以按矩阵的形式输入系统的 A, B, C 和 D 矩阵，这样就可以使用上面的第 2 个语句定义出系统的状态方程模型。如果已知系统的零极点模型，则可以输入系统的零点 zer 和极点 pol，再输入系统的增益 K，就能用第 3 个命令建立起系统的零极点模型。对单变量系统来说，注意 zer 和 pol 应该为列向量。

〖例 5.3〗 仍考虑例 5.2 中给出的双输入双输出的系统，可以使用下面的命令构造起系统的状态方程对象模型

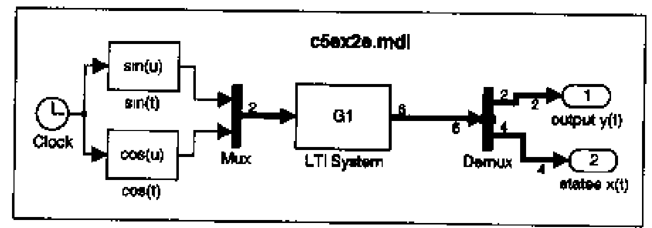
```
>> A=[2.25, -5, -1.25, -0.5;  2.25, -4.25, -1.25, -0.25;
      0.25, -0.5, -1.25,-1;  1.25, -1.75, -0.25, -0.75];
   B=[4, 6; 2, 4; 2, 2; 0, 2];
   C=[0, 0, 0, 1; 0, 2, 0, 2]; D=zeros(2,2); G=ss(A,B,C,D);
```

这样就能在 MATLAB 工作空间中建立一个 G 变量，打开 Simulink 模块库中的 Control System Toolbox，将其中的模块 LTI System (线性时不变模块) 复制到模块编辑窗口中，并在该模块中填写参数 G，则可以构造出如图 5-11 (a) 所示的框图，对该系统进行仿真则可以立即得出和前面完全一致的结果。





(a) 状态方程模型



(b) 带有状态变量检测

图 5-11　双输入双输出系统的 Simulink 模型

使用前面的思想对 C 和 D 矩阵进行增广，则可以得出下面的新系统模型 G1

```
>> C1=[C; eye(4)]; D1=[D; zeros(4,2)]; G1=ss(A,B,C1,D1);
```

从而可以构造出如图 5-11 (b) 所示的框图，除了系统输出之外，还可以同时得出系统的各个状态，对该系统进行仿真则可以立即得出和前面完全一致的结果。

〖例 5.4〗 多变量系统传递函数矩阵也可以用 LTI 对象来表示，考虑下面的 4 输入 4 输出的多变量系统模型[43]

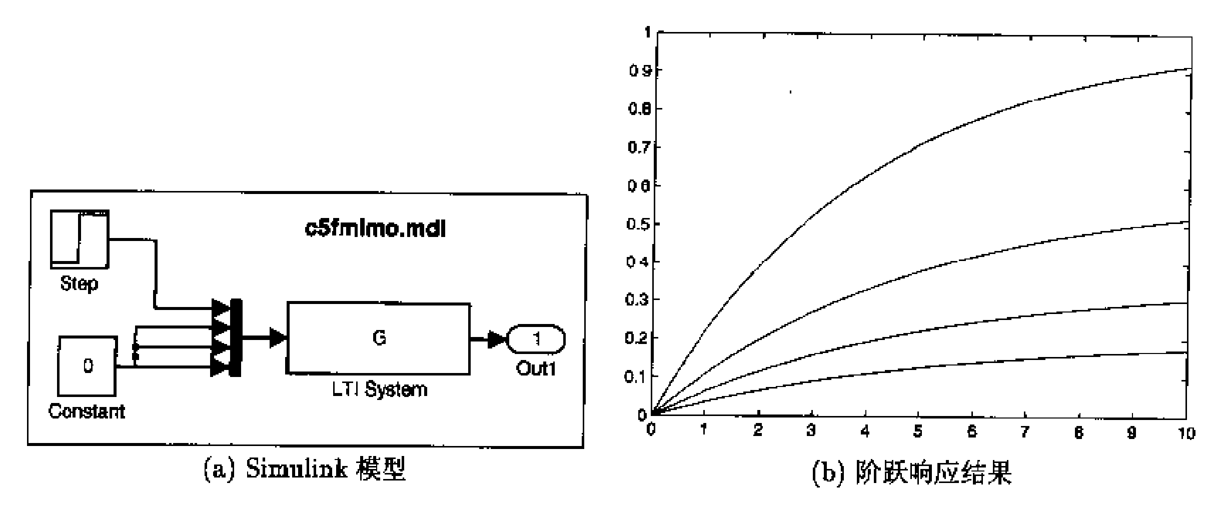
$$G(s)=\begin{bmatrix} 1/(1+4s) & 0.7/(1+5s) & 0.3/(1+5s) & 0.2/(1+5s) \\ 0.6/(1+5s) & 1/(1+4s) & 0.4/(1+5s) & 0.35/(1+5s) \\ 0.35/(1+5s) & 0.4/(1+5s) & 1/(1+4s) & 0.6/(1+5s) \\ 0.2/(1+5s) & 0.3/(1+5s) & 0.7/(1+5s) & 1/(1+4s) \end{bmatrix}$$

可以用 MATLAB 命令来输入此模型

```
>> h1=tf(1,[4 1]); h2=tf(1,[5 1]);
   h11=h1; h12=0.7*h2; h13=0.5*h2; h14=0.2*h2;
   h21=0.6*h2; h22=h1; h23=0.4*h2; h24=0.35*h2;
```

```
h31=h24; h32=h23; h33=h1; h34=h21;
h41=h14; h42=h13; h43=h12; h44=h1;
G=[h11,h12,h13,h14; h21,h22,h23,h24;
   h31,h32,h33,h34; h41,h42,h43,h44];
```

输入了系统的模型参数，则可以建立如图 5-12 (a) 所示的仿真模型，其中第 1 路输入信号为单位阶跃信号，其余信号均为 0。对整个系统进行仿真，则得出各路输出信号如图 5-12 (b) 所示。



(a) Simulink 模型 (b) 阶跃响应结果

图 5-12 多输入多输出系统的模型即仿真结果

和 MATLAB 的控制系统工具箱相比较，LTI 对象的局限性是它不能直接表示含有时间延迟的模块，而控制系统工具箱中的函数可以进行相应的运算。在 Simulink 下时间延迟应该用 Transport delay 模块单独表示，该模块可以根据需要串联于线性模块的后端。

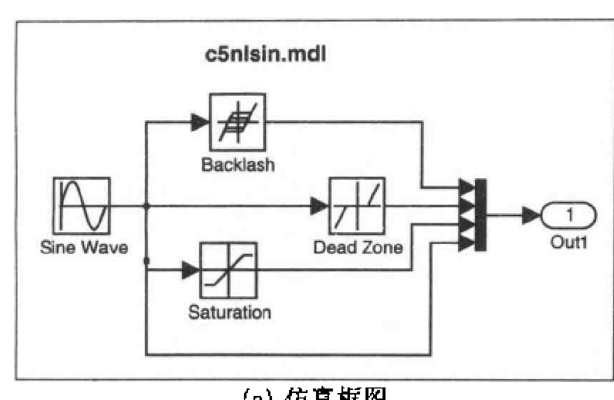
5.1.4 非线性环节与查表环节

Simulink 中提供了各种各样的非线性模块，使得非线性系统的仿真变得很容易。本节将通过几个例子来演示各种非线性环节的使用与拓展。

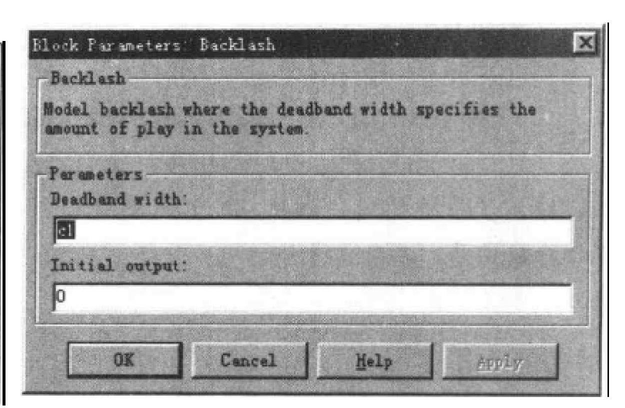
〖例 5.5〗 在正弦信号激励下，经过几个常用的非线性模块，如磁滞回环、死区特性和饱和特性等，试观察输出信号的曲线形状。出于这个目的，可以容易地构造出如图 5-13 (a) 所示的系统框图，再设置正弦信号的幅值 (正弦信号模块中的 Amplitude 属性) 为 2，并将各个非线性环节的参数设置为 c1。首先设置 c1=0.5，对整个系统进行仿真则将得出 tout 和 yout 两个变量，用下面的命令可以绘制出正弦信号经过非线性环节后的结果，如图 5-14 所示。

```
>> subplot(221), plot(tout,yout(:,4))
   subplot(222), plot(tout,yout(:,1))
   subplot(223), plot(tout,yout(:,2))
   subplot(224), plot(tout,yout(:,3))
```

再改变 c1 的值为 1，进行仿真并重复上面的过程，则将得出如图 5-15 所示的仿真结果。可以看出，利用 Simulink 强大的功能就能轻易地解决提出的问题。



(a) 仿真框图



(b) 非线性模块参数设置

图 5-13　非线性模块仿真框图

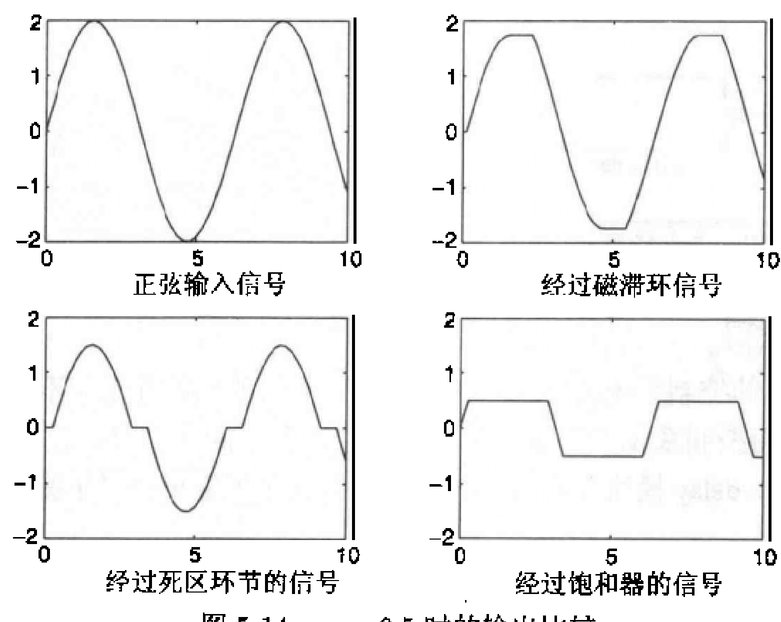


图 5-14　$c_1 = 0.5$ 时的输出比较

从 Simulink 中提供的非线性模块组看可能会觉得，该模块组只提供了有限几种静态非线性模块，不一定能包含所需要的所有模块。事实上，很多非线性模块是可以由该模块组中给出的模块搭建而成，例如，既带有死区也带有饱和的非线性环节可以由一个死区非线性环节和一个饱和非线性环节串联而成，如图 5-16 所示[3]。

另外，由于 Simulink 的 Functions and Tables 模块组中提供了一维查表的模块 (Look-up Table 模块)，所以可以建立起所有无记忆的分段线性的非线性环节。双击该图标将得出如图 5-17 (a) 所示的对话框，只需在该对话框中填写出该模块的转折点的坐标，就能建立起所需的非线性环节。

〖例 5.6〗 考虑图 5-16 中给出的非线性环节，可以得出该模块的转折点坐标为$(-4,-2)$, $(-3,-2)$, $(-1,0)$, $(1,0)$, $(3,2)$, $(4,2)$，这样可以在该对话框的 Vector of input values (输入数据向量) 编辑框中填写 [-4,-3,-1,1,3,4]，在 Vector of output values (输出数据向量) 编辑框中填写 [-2,-2,0,0,2,2]，

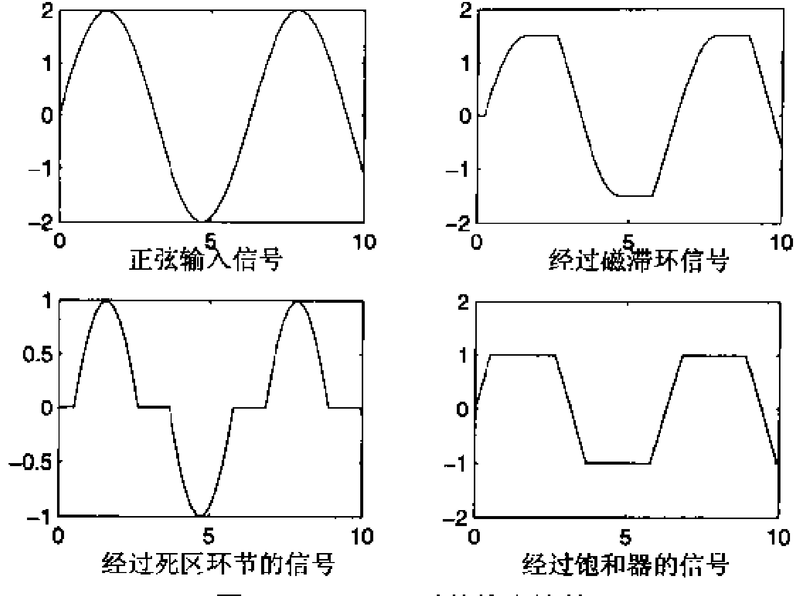


图 5-15 $c_1 = 1$ 时的输出比较

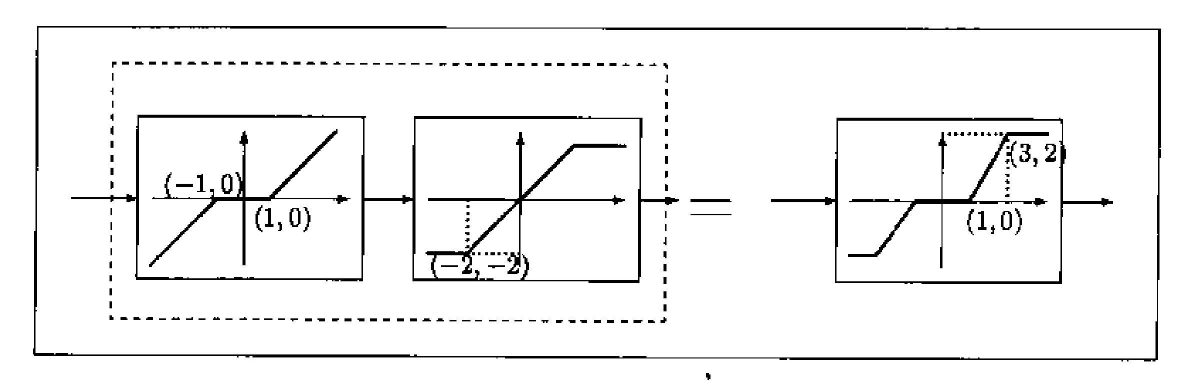
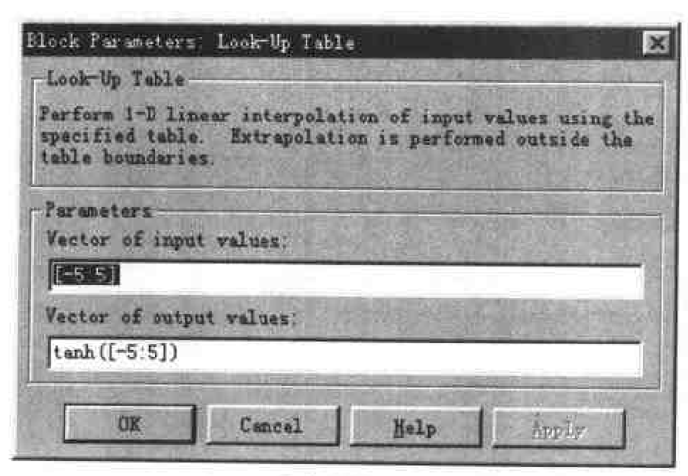


图 5-16 复杂非线性环节的等效搭建





(a) 一维查表模块对话框

Look-Up Table

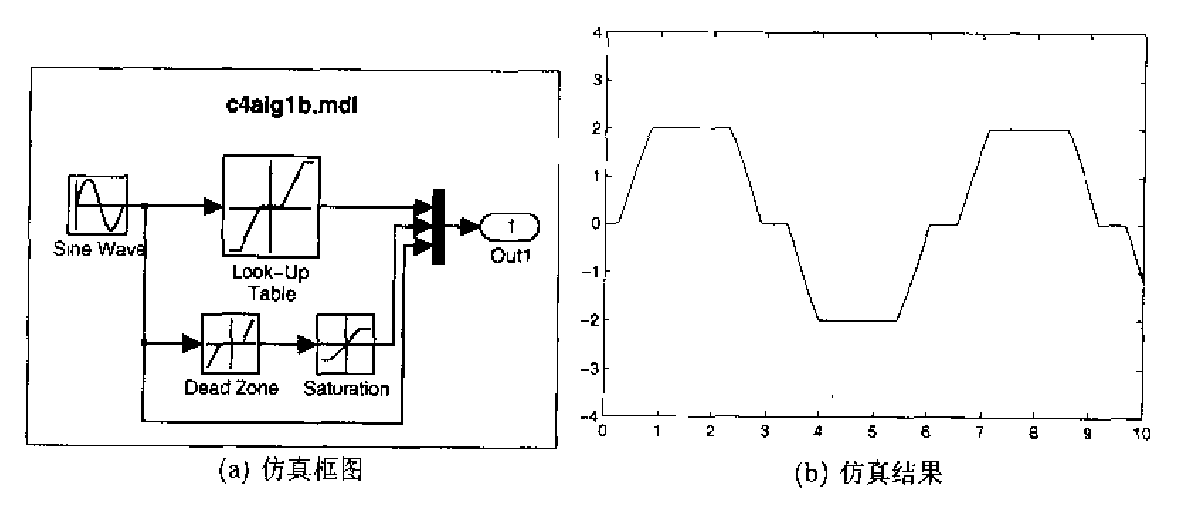
(b) 查表模块图标

图 5-17 一维查表模块

则可以建立起所需的非线性环节，这时该模块图标将自动变成如图 5-17 (b) 所示的形式。

掌握了这样的方法，就可以依赖查表模块轻易地建立起任意无记忆的非线性环节了。可以将这样的模块建立起仿真模型，如图 5-18 (a) 所示。在该图中还绘制了按图 5-16 方式搭建的非线性模块，其中死区模块的参数填写为 −1 和 1，饱和非线性模块的参数填写为 −2 和 2。

通过它来观察正弦信号经过该环节后的输出效果，如图 5-18 (b) 所示，且两种方式得出的输出完全一致。这里，假设输入信号的幅值为 4。为方便比较起见，还同时用虚线绘制出了输入的正弦信号，这是通过修改曲线属性实现的。



(a) 仿真框图　　(b) 仿真结果

图 5-18　非线性模块仿真框图

〖例 5.7〗 由给出的例子可以看出，任何的单值非线性函数均可以采取这样的方式来建立或近似，但如果非线性中存在回环或多值属性，则简单地采用这样的方法是不能构造的，解决这类问题则需要使用开关模块。

假设想构造一个如图 5-19 所示的回环模块。可以看出，该特性不是单值的，该模块中输入在

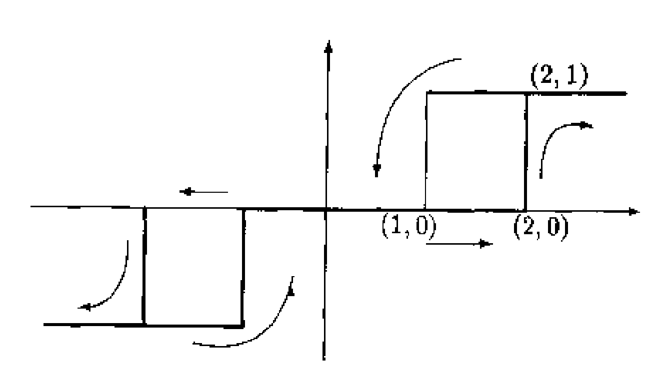
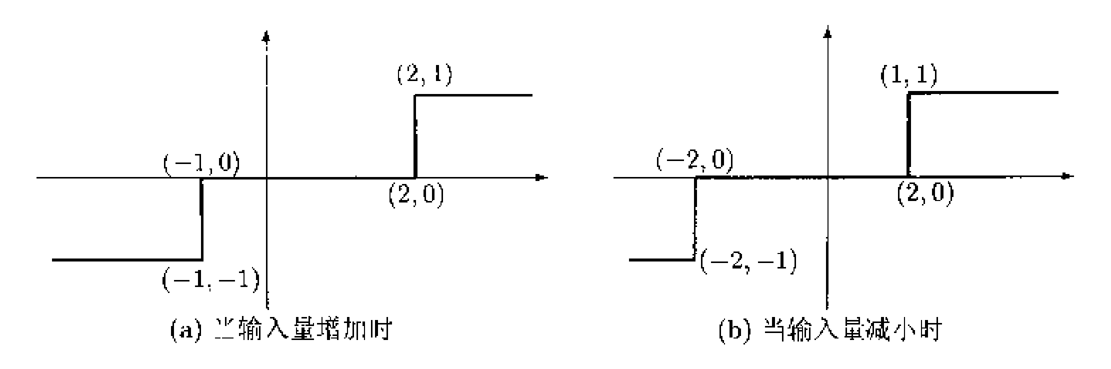


图 5-19　给定的回环函数表示

增加时走一条折线，减小时走另一条折线。将这个非线性函数分解成如图 5-20 所示的单值函数，当然这个单值函数是有条件的，它区分输入信号上升还是下降。

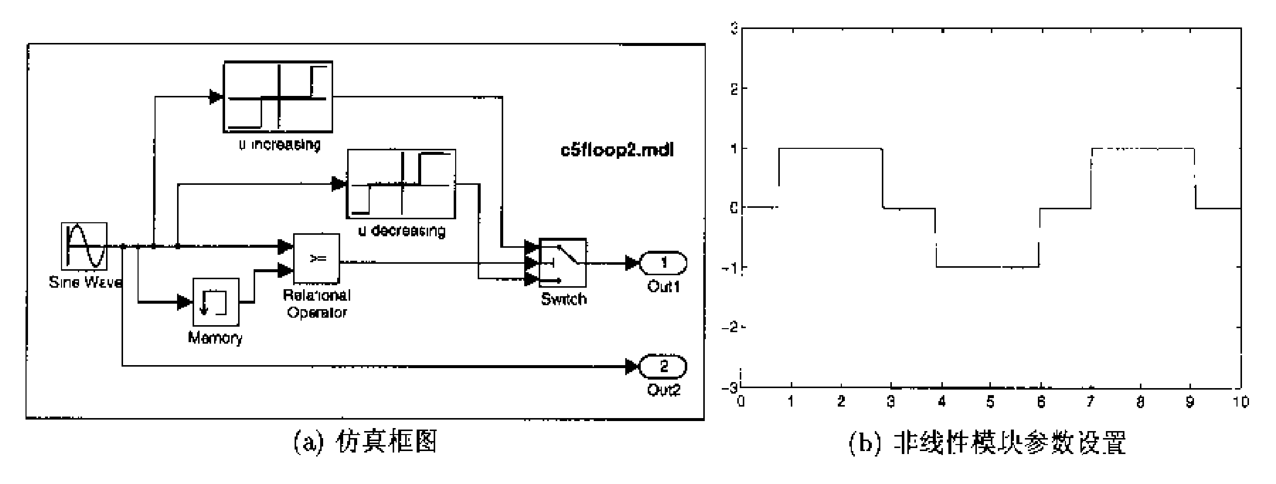
Simulink 的连续模块组中提供了一个 Memory (记忆) 模块，该模块记忆前一个计算步长上的



(a) 当输入量增加时　　(b) 当输入量减小时

图 5-20 回环函数分解为单值函数

信号值，所以可以按照图 5-21 (a) 中所示的格式构造一个 Simulink 模型。在该框图中使用了一个



(a) 仿真框图　　(b) 非线性模块参数设置

图 5-21 正弦信号通过回环函数的仿真结果

比较符号来比较当前的输入信号与上一步输入信号的大小，其输出是逻辑变量，在上升时输出的值为 1，下降时的值为 0。由该信号可以控制后面的开关模块，设开关模块的阈值 (Threshold) 为 0.5，则当输入信号为上升时由开关上面的通路计算整个系统的输出，而下降时由下面的通路计算输出。

两个查表模块的输入输出分别为

$x_1 = [-3, -1, -1+\epsilon, 2, 2+\epsilon, 3]$，$y_1 = [-1, -1, 0, 0, 1, 1]$

$x_2 = [-3, -2, -2+\epsilon, 1, 1+\epsilon, 3]$，$y_2 = [-1, -1, 0, 0, 1, 1]$

其中，ϵ 可以取一个很小的数值，例如可以取 MATLAB 保留的常数 eps。设输入正弦信号的幅值为 3，则可以得出如图 5-21 (b) 所示的仿真结果，其中虚线表示的仍为输入的正弦信号。

修改非线性回环函数的结构，使其如图 5-22 所示，则仍可以利用前面建立的 Simulink 模型，只需修改两个查表函数成

$x_1 = [-3, -2, -1, 2, 3, 4]$，$y_1 = [-1, -1, 0, 0, 1, 1]$

$x_2 = [-3, -2, -1, 1, 2, 3]$，$y_2 = [-1, -1, 0, 0, 1, 1]$

从而立即就能得出整个系统的框图，如图 5-23 (a) 所示。对该系统框图进行仿真，则能得出如图

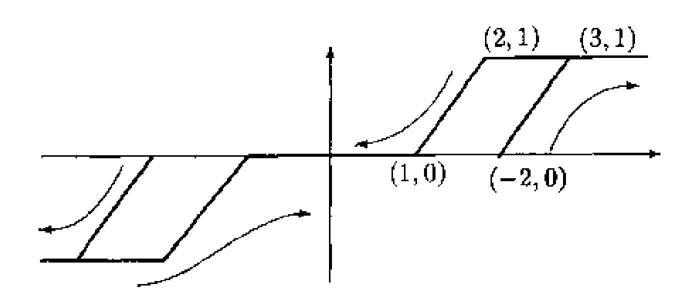
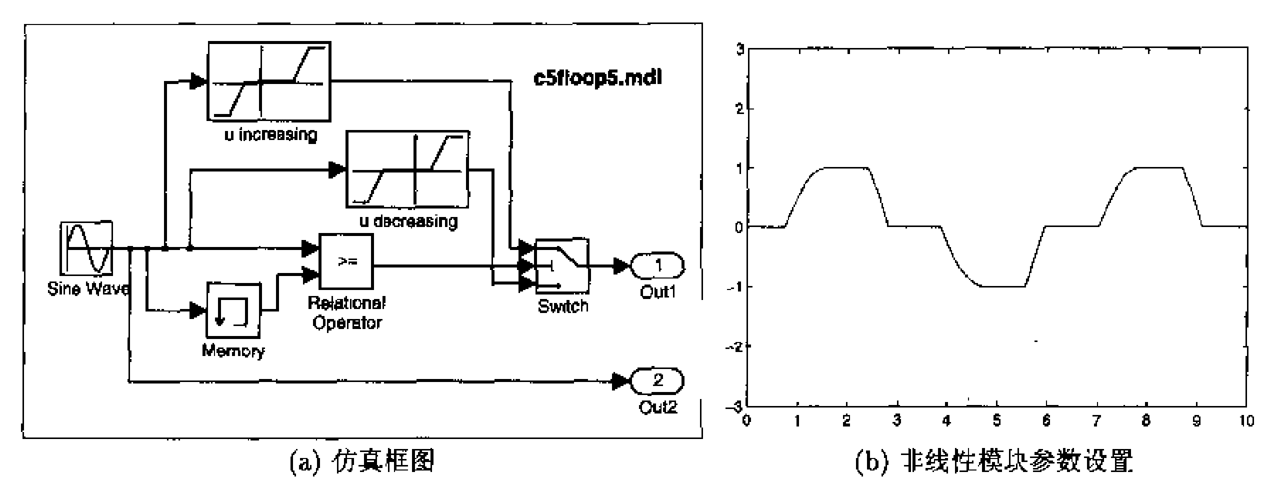


图 5-22 新的回环函数表示

5-23 (b) 所示的输出曲线。



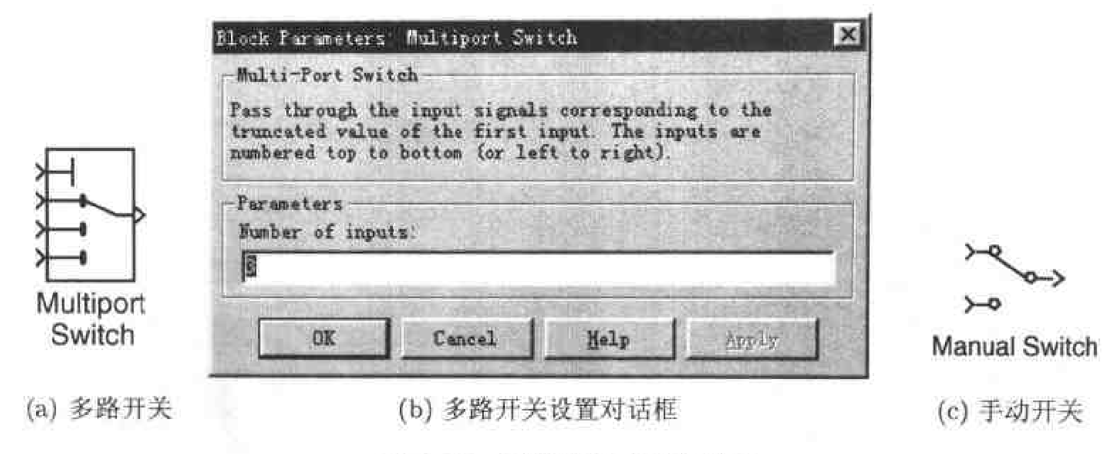
(a) 仿真框图 (b) 非线性模块参数设置

图 5-23 正弦信号通过新回环函数的仿真结果

Simulink 在非线性模块组中还提供了多路开关 (Multiport Switch) 和手动开关 (Manual Switch)，前者如图 5-24 (a) 所示，其最上面的是控制信号，该信号等于 $1,2,\cdots,n$ 时分别对应于各路控制信号，若不等于这些数值则将给出错误信息。可以双击该模块，在得出的如图 5-24 (b) 所示对话框中选定多路开关输入信号的路数。

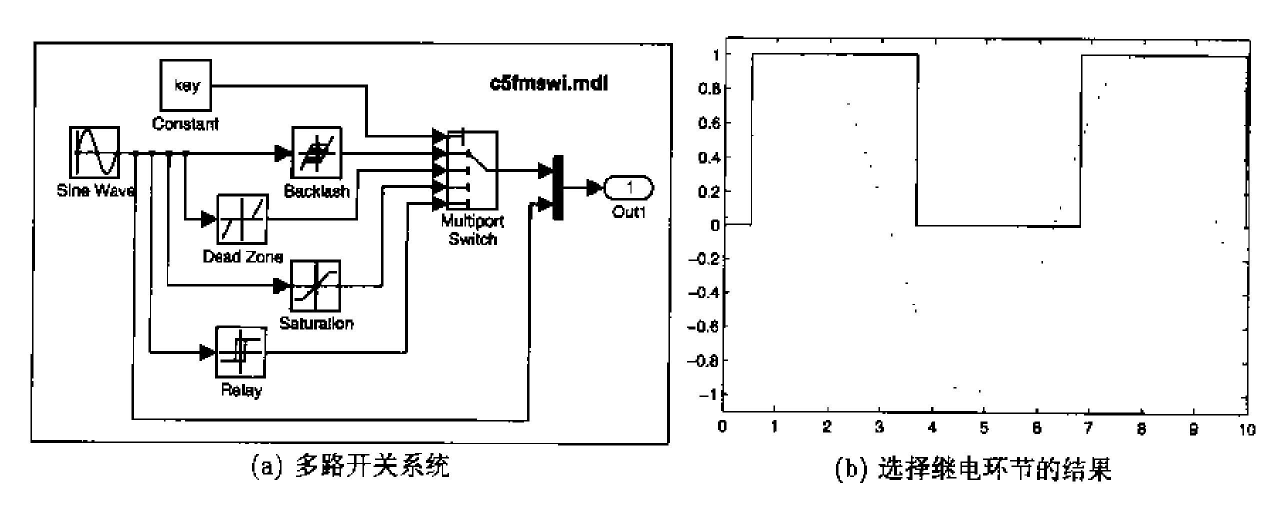
手动开关也是 Simulink 的非线性模块组中很具特色的模块，如图 5-24 (c) 所示，该模块在用户双击时切换状态。在很多场合都可以使用这样的开关来模拟实际对象，如自整定 PID 控制器的仿真中可以采用手动开关来切换系统的运行状态，从而到达自整定的目的。

〖例 5.8〗 再考虑例 5.5 中的问题，可以采用多路开关的方式来选择非线性环节，最终直接返回计算结果。根据这样的想法，可以绘制出如图 5-25 (a) 所示的 Simulink 框图。在该框图中设置了一个外部输入常数 key，在 MATLAB 工作空间中可以定义其值，例如选择 key=4。另外，可以将继电环节参数设置为 0.5 和 −0.5，对这样的系统进行仿真，则将得出如图 5-25 (b) 所示的仿真曲线，同样这里将输入信号设置为虚线。将 key 设置成其他的值则可以选择各路非线性环节，得出所需的输出曲线。



(a) 多路开关　　(b) 多路开关设置对话框　　(c) 手动开关

图 5-24　多路开关和手动开关



(a) 多路开关系统　　(b) 选择继电环节的结果

图 5-25　多路开关及仿真结果

Simulink 不但提供了一维表格查询的模块，还提供了二维或多维查表模块。提供的多维查表使用线性插值的算法求出输出。下面以二维查表模块为例演示多维查表的原理与应用。

〖例 5.9〗 考虑例 2.23 中给出的二维函数

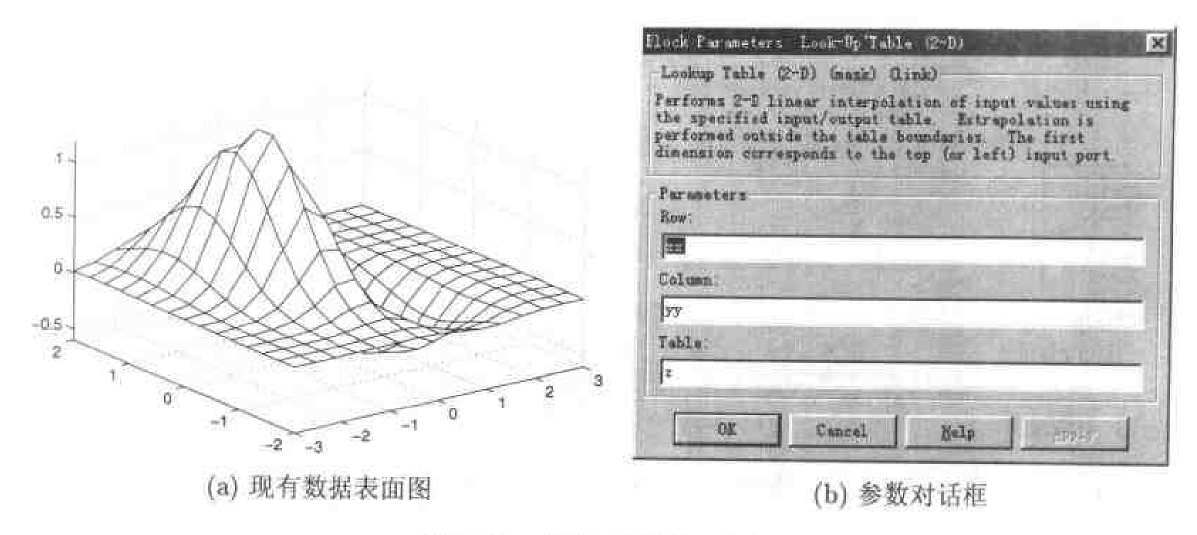
$$z=f(x,y)=(x^2-2x)\mathrm{e}^{-x^2-y^2-xy}$$

假设能构造较稀疏的 x-y 平面上的网格点，并可以计算出在这些点上的高度值

```
>> xx=linspace(-3,3,15); yy=linspace(-2,2,15);
   [x,y]=meshgrid(xx,yy);
   z=(x.^2-2*x).*exp(-x.^2-y.^2-x.*y);
   mesh(xx,yy,z); axis([-3,3,-2,2,-0.6,1.2])
```

这样已知点的数据就可以用 xx, yy 和 z 表示出来，如图 5-26 (a) 所示。双击二维查表模块图标得出的对话框，可以将这三个向量分别填入相应的栏目，如图 5-26 (b) 所示。

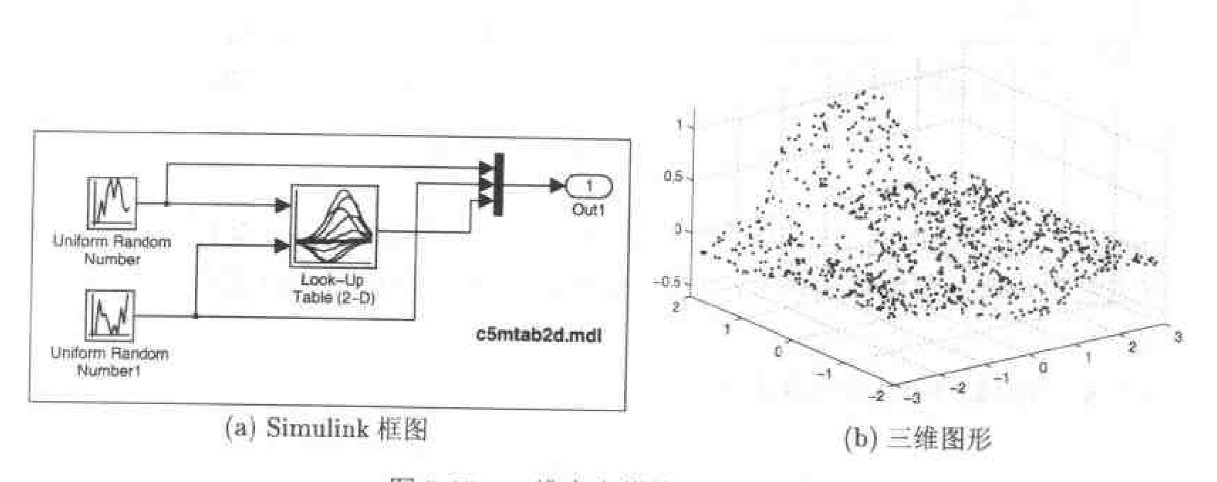
现在试图用均匀分布随机数的方式生成一组 x 和一组 y 变量作为输入信号，输入到二维查表



(a) 现有数据表面图　　(b) 参数对话框

图 5-26　多维查表模块使用

模块，就可以构造出如图 5-27 (a) 所示的 Simulink 框图。注意，在两个随机数发生模块中，伪随机数的种子不能选择相同的值，否则两路输入信号将是一致的，不能很好演示这里的例子。例如可以将第 2 路随机数种子选择为 31242240。另外，假设两个随机数发生模块的范围分别为 $(-3,3)$ 和 $(-2,2)$。在仿真中如果选择定步长为 0.001，并设定终止仿真时间为 10，则可以用仿真的方法



(a) Simulink 框图　　(b) 三维图形

图 5-27　二维查表模块设置与仿真

计算出 10,000 个点，利用这些点用描点的方式可以就绘制出三维图，如图 5-27 (b) 所示。

```
>> plot3(yout(:,2),yout(:,1),yout(:,3),'.'); axis([-3,3,-2,2,-0.6,1.2])
```

可以看出其形状类似于已知数据得出的图形。遗憾的是，这个模块采用的是线性插值的方法，所以在精度上较差，一般不能得到平滑的效果。

Simulink 还允许进行多维查表运算，所使用的方法是线性插值。输入输出数据给定还是比较麻烦的，因为它要求给出网格数据，而一般实测的数据不是网格型的。

5.1.5 微分代数方程的 Simulink 建模与求解

在第 3.4.6 节中介绍过微分代数方程的概念，并对给出的例子 (例 3.37) 进行了 MATLAB 求解，其实用 Simulink 也能构造出微分代数方程的仿真模型，因为在 Simulink 的数学函数模块组中，给出了代数约束模块，则可以用该模块构造代数方程。

〖例 5.10〗 考虑例 3.37 中给出的微分代数方程模型，为方便起见，在这里重新写出原始问题

$$\begin{cases} \dot{x}_1 = -0.2x_1 + x_2x_3 + 0.3x_1x_2 \\ \dot{x}_2 = 2x_1x_2 - 5x_2x_3 - 2x_2^2 \\ x_1 + x_2 + x_3 - 1 = 0 \end{cases}$$

并已知初始条件为 $x_1(0) = 0.8, x_2(0) = x_3(0) = 0.1$。因为两个信号 $x_1(t)$ 和 $x_2(t)$ 可以设定为积分器的输出，所以可以将它们视为已知信号，故最后的方程可以认为是 $f(x_3) = x_1+x_2+x_3-1 = 0$，用代数约束模块构造出次代数方程的模型表示，其输出信号即解出的 $x_3(t)$ 信号。这样就能构造出如图 5-28 所示的 Simulink 模型，其中将两个积分器的初值分别设置为 0.8, 0.1，并将代数约束的初始值设置为 0.1。

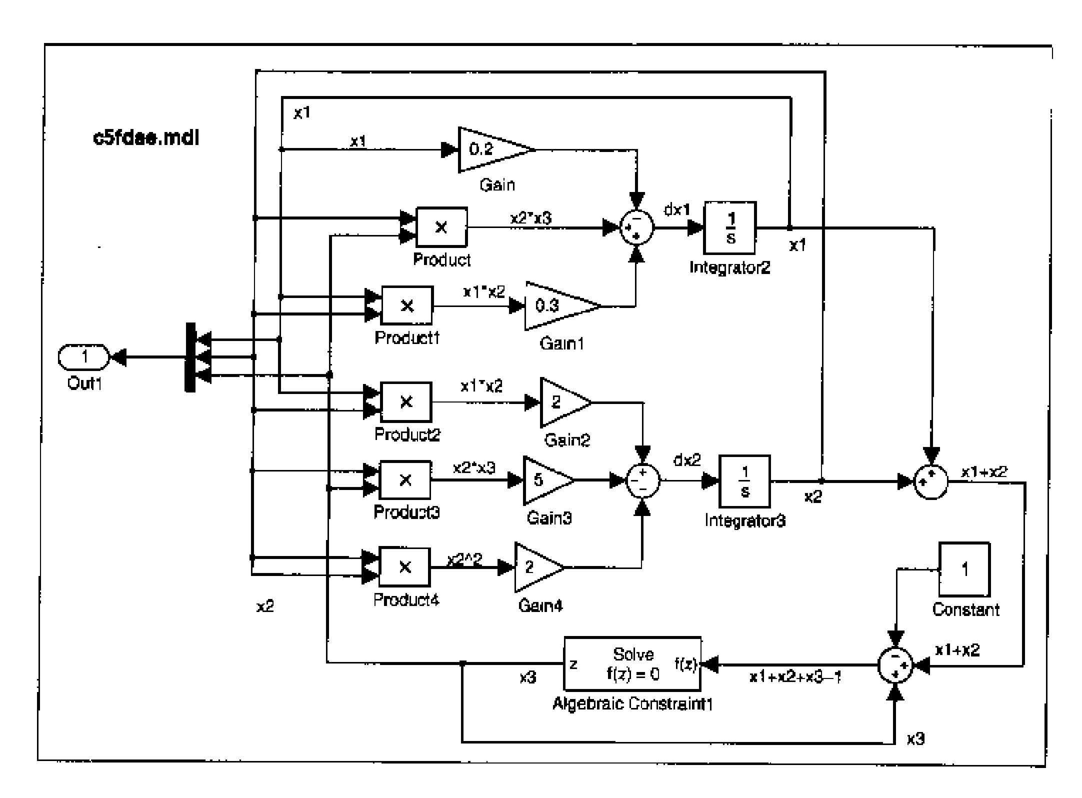


图 5-28 微分代数方程的仿真框图

正如例 3.37 所指出，对这样的微分代数方程直接在默认参数内采用四阶五级 Runge-Kutta 算法得出的结果不是很可靠，所以在仿真参数中，可以选择 `ode15s` 算法，或降低 `ode45` 算法中的相对误差要求 (如选择其为 `1e-4`)。

5.2 输出模块库

输出环节是 Simulink 仿真中重要的一环，因为仿真的目的是从给定的模型得到某种计算结果，而结果是应该以所需的形式返回给用户的。Simulink 仿真结果有这样几种途径输出出来：

- **示波器输出** 前面介绍过，示波器输出可以有两种方式，普通的示波器和相轨迹示波器，前者显示信号的时间曲线，后者显示两路输入信号之间的关系。示波器输出提供了一种便捷的方式，亦即无需在仿真后给任何附加命令就能得到仿真结果，在默认的情况下这样的方式不能将结果数据返回到 MATLAB 的工作空间进行后处理，无法利用 MATLAB 强大的图形绘制功能。
- **直接数据显示** 输出池模块组中提供了一个数值显示模块 (Display)，它将以数字显示的形式连到其上的信号，该模块可以同时显示多路信号。值得指出的是，由于仿真是非实时的，所以对一般问题来说仿真过程过快，所以不适合连接数字显示模块观察输出信号。
- **输出端子** 前面的例子中一直采用这样的输出方式，这样的方式在仿真结束后在 MATLAB 的工作空间中自动生成两个变量，`tout` 和 `yout`，分别返回时间向量和各个输出端子的仿真结果。这样返回的变量可以方便地在 MATLAB 环境下进行后处理，所以采用这样的方式输出仿真结果将是很合适的。
- **返回工作空间** 将输出池中的 To workspace 模块直接连接到要观测的信号上就可以将该信号返回到 MATLAB 工作空间进行后处理，用户可以自由地选择想显示的 MATLAB 变量名，自动生成的时间变量仍为 `tout`。
- **文件输出** 将仿真结果直接存到数据文件中。
- **表盘与量计显示** Simulink 利用一组 ActiveX 部件，将仿真的结果采用表盘或量计的形式显示出来，这需要事先安装 Dials and Gauges 模块集。用表盘或量计显示的方式输出仿真结果类似于实际过程控制现场见到的显示仪表。
- **数字信号处理、分析** 可以在一些信号的后面直接连数字信号处理模块，以便获得信号的相关函数、功率谱密度、快速 Fourier 变换等结果。

值得指出的是，每路输出信号允许同时连接多个输出模块，例如，一个信号既可以连示波器，也可以同时连输出端子和表盘显示，另外还可以连接一个工作空间存储数据模块将结果写入工作空间，如图 5-29 所示。

5.2.1 一般输出模块库

5.2.1.1 示波器与端子输出

Simulink 的示波器输出方式在前面已经演示过，比如在 Van der Pol 方程演示 (例 4.1) 中，曾经在同一个示波器上显示过两条曲线，这是通过将想显示的信号经过 Mux 模块进行向量化而实现的。事实上示波器也可以单独接受多路信号，单击示波器工具栏上

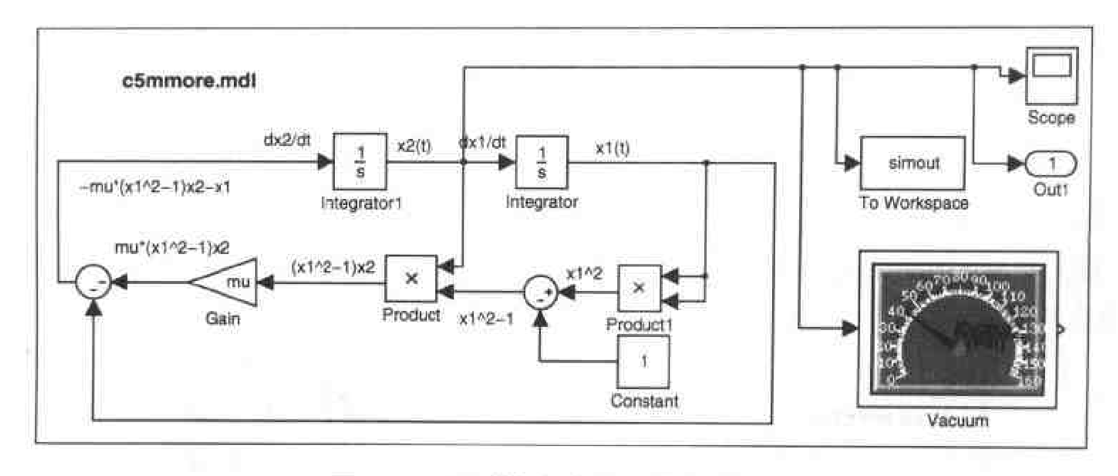
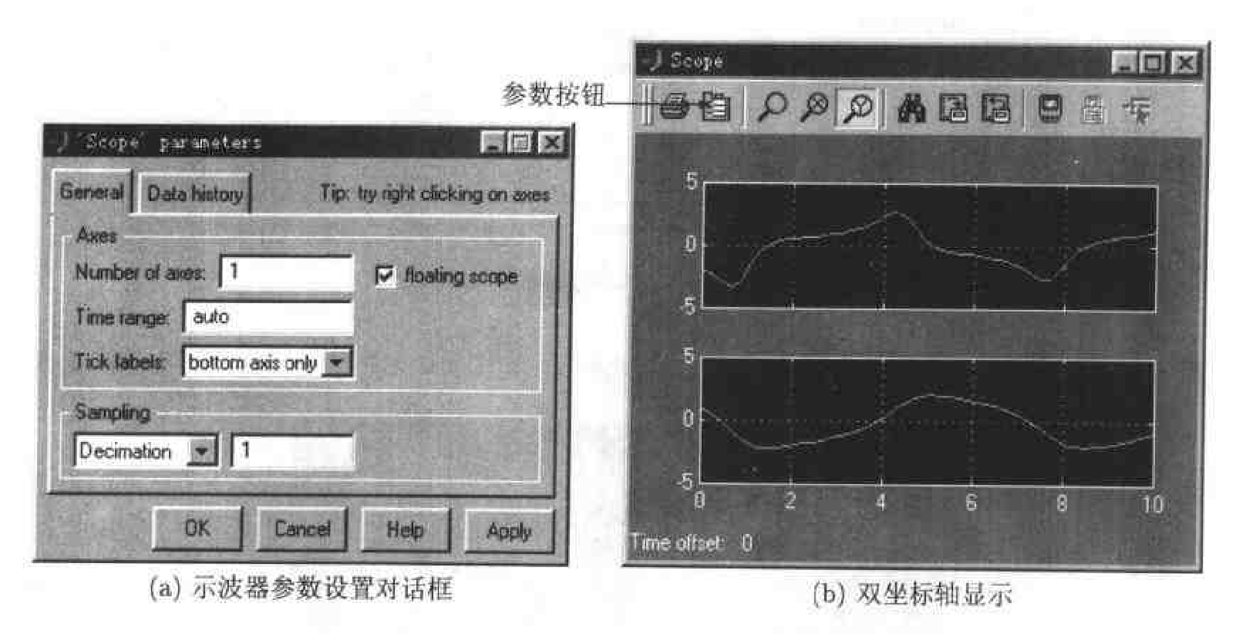


图 5-29 同时接多个输出模块的框图



(a) 示波器参数设置对话框

(b) 双坐标轴显示

图 5-30 示波器选项及多坐标轴显示

左数第 2 个按钮 (Parameters 按钮)，将打开一个如图 5-30 (a) 所示的对话框，在 Number of axes (坐标轴数目) 栏目下输入 2，则该模块会自动生成两个输入端口，将端口分别连到 Van der Pol 方程模型的 x_1 和 x_2 信号上，再进行仿真，则将得出如图 5-30 (b) 所示的示波器输出结果。当然，还可以重新调整示波器窗口的形状，使得它有更好的可读性。

输出端子方式是一种实用的信号输出方式，但应该注意，该方式和仿真参数对话框中的 Workspace I/O 标签 (图 4-27) 是密切相关的，亦即在 Save to workspace 栏目内必须选中 tout 和 yout 两个选项，否则将不能将这两个变量返回。

另外值得注意的是该对话框中的 Limit data points to last (存储最近数据点数) 的复选框处于被选中状态，这将自动返回最新的 1000 个仿真点，如果实际仿真点数超过这个数值，而又想得到仿真的全貌，则取消此框的选中状态。

5.2.1.2 工作空间及文件输出

To workspace 输出模块曾经是 Simulink 中最重要的输出模块，随着输出端子模块功能的日益增强，现在在仿真中一般不再依赖该模块了。这里仍然以 Van der Pol 模型为例来介绍该模块的使用。

〖例 5.11〗 对图 4-35 所示的模型进行改装，将原有的示波器替换成 To workspace 模块，则可以得出如图 5-31 所示的模型。

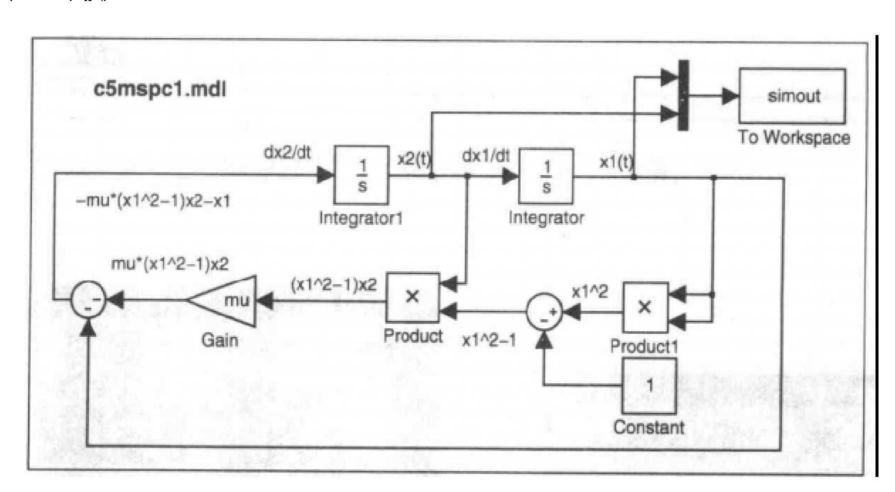


图 5-31 带有 To workspace 模块的模型

双击该输出模块，将得出如图 5-32 所示的对话框，该对话框允许用户为返回的变量选择名字

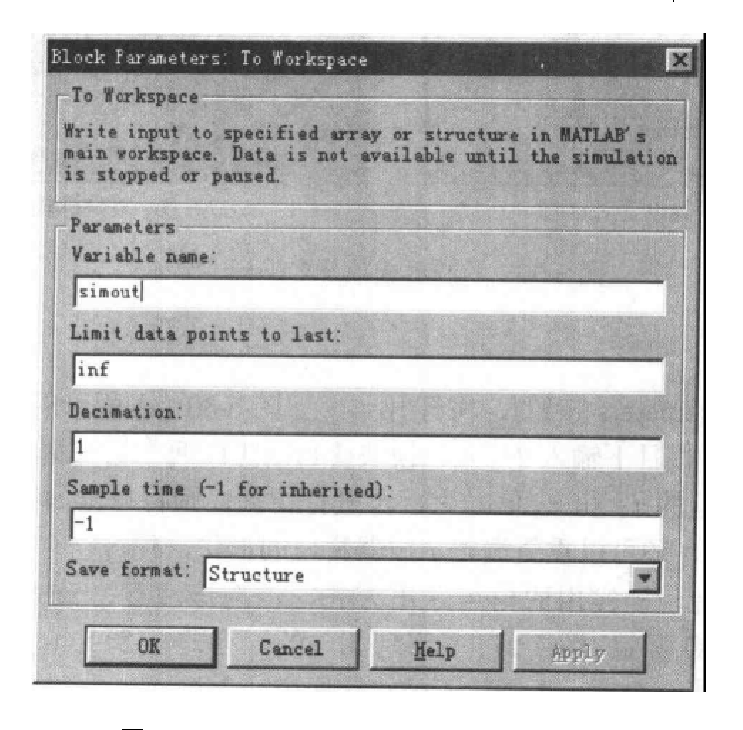


图 5-32 To workspace 参数设置对话框

(Variable name 栏目)，另外 Save format (存储数据格式) 栏目允许用户选择 Structure (结构体，默

认)、Array (矩阵格式) 和 Structure with time (带有时间变量的结构体) 三种格式。

在默认的数据格式下，simout 有三个成员变量

```
>> simout
simout =
        time: []
     signals: [1x1 struct]
   blockName: 'c5mspc1/To Workspace'
```

其中 time 变量为空矩阵，如果选择了 Structure with time 格式，则该变量将为仿真中的时间点构成的列向量。signals 成员变量本身也是一个结构体，其结构为

```
>> simout.signals
ans =
       values: [52x2 double]
   dimensions: 2
        label: ''
```

所以如果需要仿真结果数据，必须使用 simout.signals.values 语句才能获得。这时若需要绘制输出曲线，则需要给出下面的命令

```
>> plot(tout,simout.signals.values)
```

可见，这样的返回格式还是显得过于烦琐，虽然它除了数据本身以外还能同时返回其他信息。如果选择了 Array 选项，若想获得数据则用 simout 本身即可。

其实在输出端子设置中也存在这三种数据格式，见图 4-27 中给出的对话框，只不过该对话框的默认值设置成了 Array 格式，所以使用起来没有太多的麻烦。

如果使用输出池中的 To File 模块，双击该模块可以得出如图 5-33 所示的对话框，则

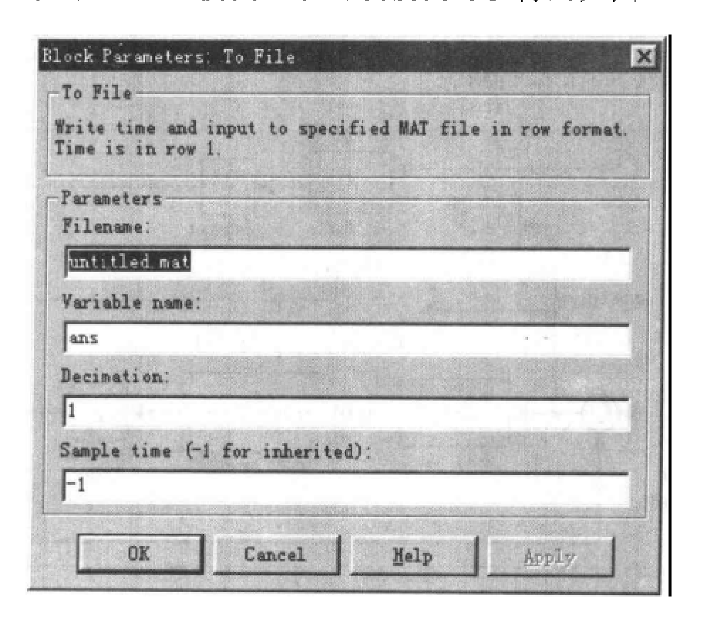


图 5-33 To File 模块参数设置对话框

将自动以 Mat 文件的格式存储数据到指定的文件中，文件名可以在其中的 File name 栏目

填写。其数据格式为：第 1 行为时间向量，第 2 行为第 1 路信号，以后各行依此类推。以这样方式存储的文件可以用 From File 模块直接使用，也可以在 MATLAB 环境中用 load 命令将其读入工作空间，数据在 MATLAB 工作空间中的变量名可以由 Variable name 栏目填写。

5.2.1.3 输出模块应用举例

这里将通过几个例子演示 Simulink 不同的输出方法。在前一个例子中将演示和时间有关的变量输出显示方法，第二个例子将演示 STOP 模块在仿真过程控制中的应用。

【例 5.12】 在控制系统分析中，经常需要获得某个信号的某些时域指标，如 ITAE (integral of time weighted absolute errors) 指标，其定义为

$$f(e) = \int_0^t \tau \mid e(\tau) \mid \mathrm{d}\tau \tag{5.1}$$

其中 $e(\cdot)$ 为系统的误差信号。另外 ISE (integral of squared error) 也是很常用的指标，其定义为

$$g(e) = \int_0^t e^2(\tau)\mathrm{d}\tau \tag{5.2}$$

现在考虑图 5-34 中给出的控制系统模型，用 Simulink 可以建立起其框图，并依照式 (5.1) 和

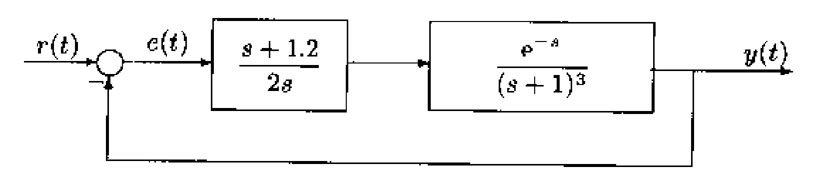


图 5-34 控制系统框图

(5.2) 建立其两个指标函数，如图 5-35 所示。

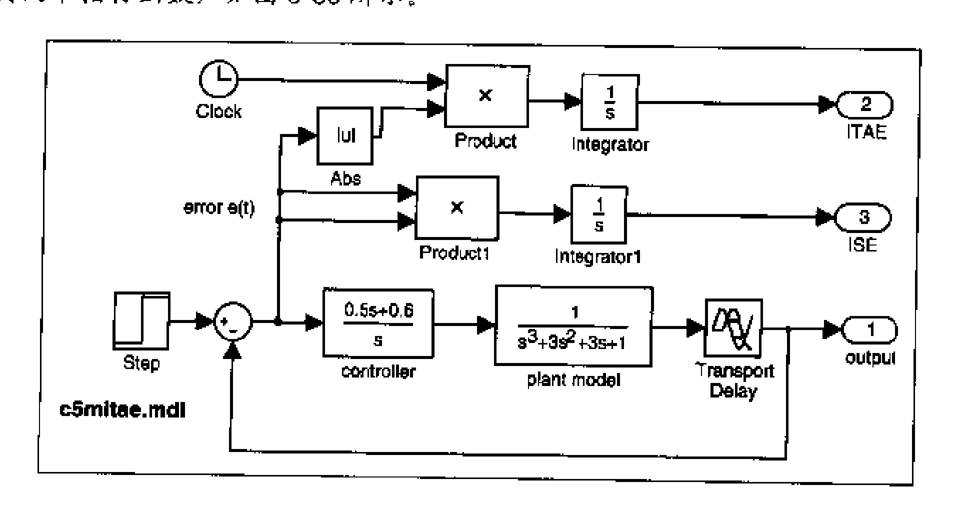
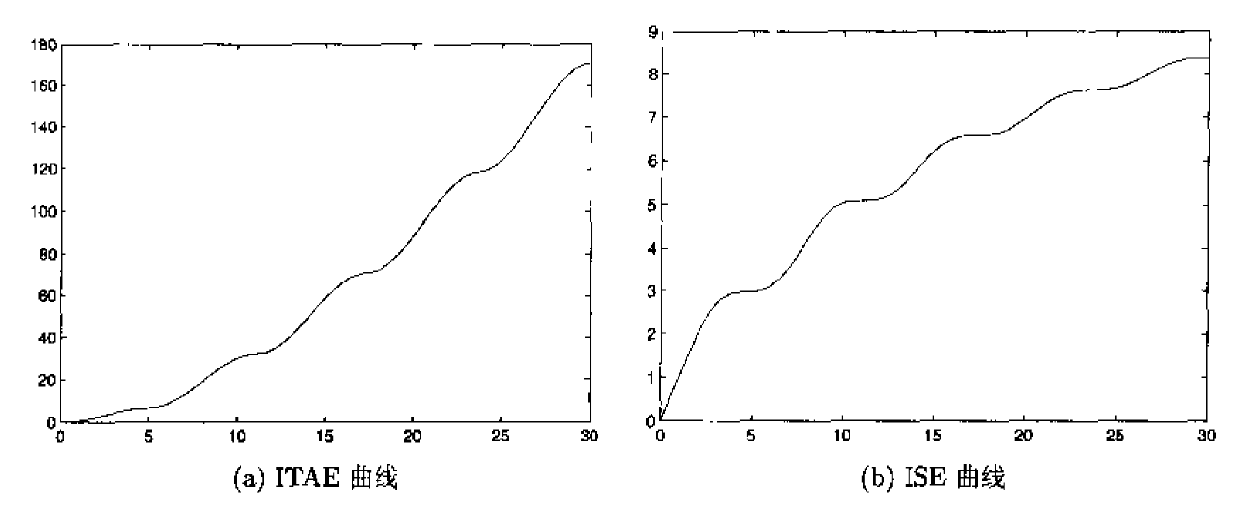


图 5-35 带有 ITAE 和 ISE 的 Simulink 框图

构造出模型后，可以设置终止仿真时间为 30，这样就可以进行整个系统的仿真，仿真结束后则可以用下面语句绘制出两种误差准则曲线，如图 5-36 所示。

```
>> plot(tout,yout(:,2)); % 绘制 ITAE 指标
   figure; plot(tout,yout(:,3)) % 打开新图形窗口，并绘制 ISE 指标
```



(a) ITAE 曲线　　(b) ISE 曲线

图 5-36 控制系统的误差准则曲线

〖例 5.13〗 考虑前面的 PI 控制的例子，若人为地选择仿真终止时间为 30，则将得出如图 5-37 所示的曲线，从该曲线中可见，输出信号在选定的时间区域内并未收敛到一个较小的范围内，用控

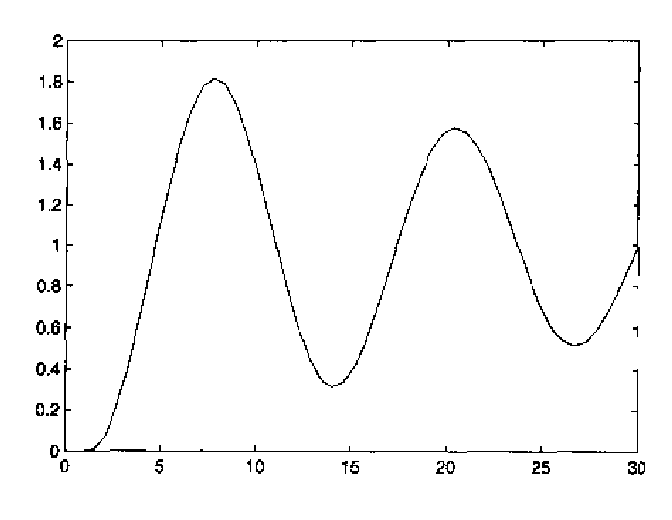


图 5-37 PI 过程控制系统的输出信号

制理论中常用的语言来说，尚未达到系统的调节时间。这将提出一个问题：如何自动地选择一个时间范围，使得输出被调节到理想的结果。

在这个例子中将演示 STOP 模块的应用，该模块在其输入信号非 0 时将自动终止仿真过程。可以按图 5-38 所示的方式搭建出 Simulink 模型。在该模型的上部是用来判定进入调节区域足够长时间的。具体的解释是：取误差信号的绝对值，如果其值大于 0.02 则返回 0，否则返回 1。将该信号乘以 -1，得出 $x_1(t)$，再对之取积分，并将积分器设置一个复位信号，让 x_1 信号的上升沿来触发积分器的复位信号，其对话框设置如图 5-39 (a) 所示。这时令该触发信号为被积信号，则积分器的输出在 $|e(t)| < 0.02$ 满足时将为一下降直线，该信号 $x_2(t)$ 的值越小，则说明满足该条件

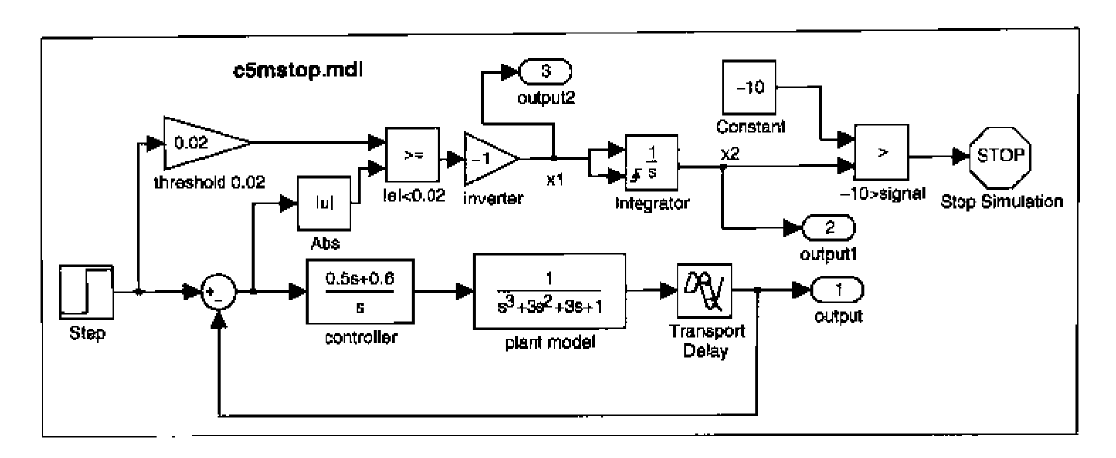
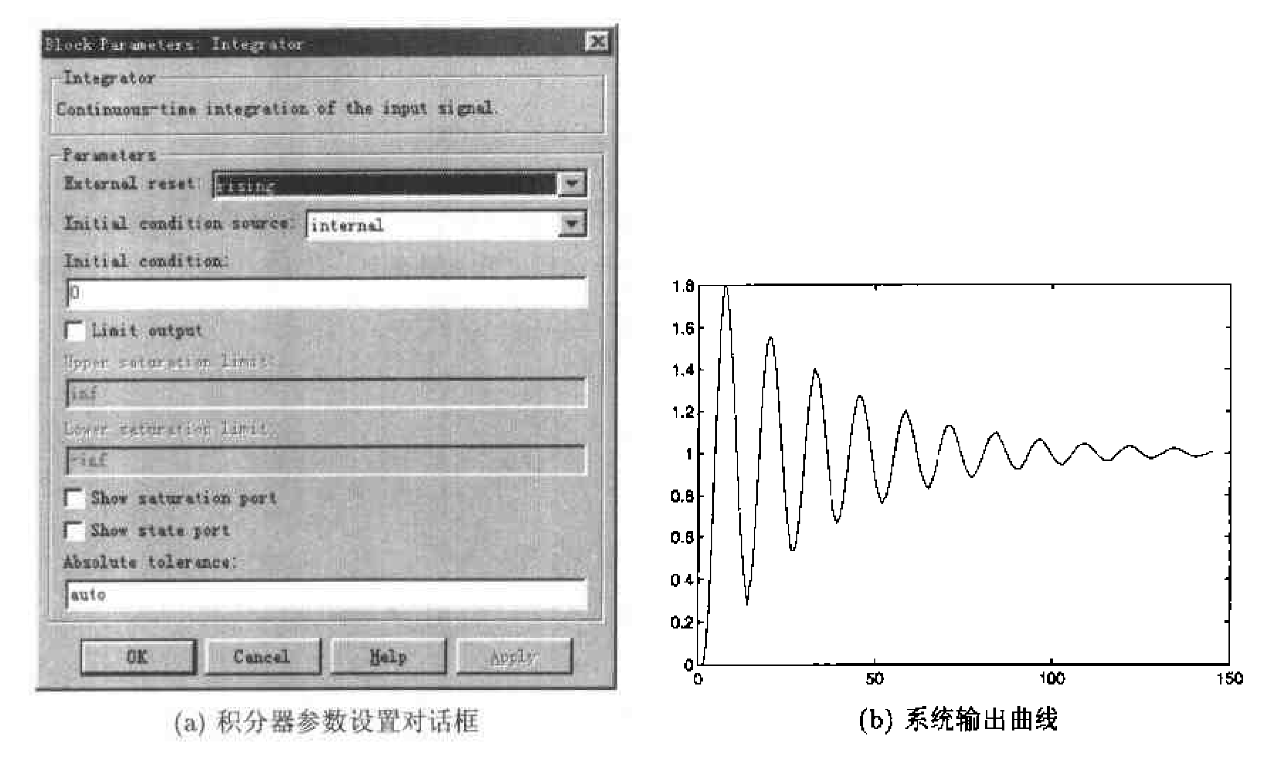


图 5-38 Simulink 框图



(a) 积分器参数设置对话框　　(b) 系统输出曲线

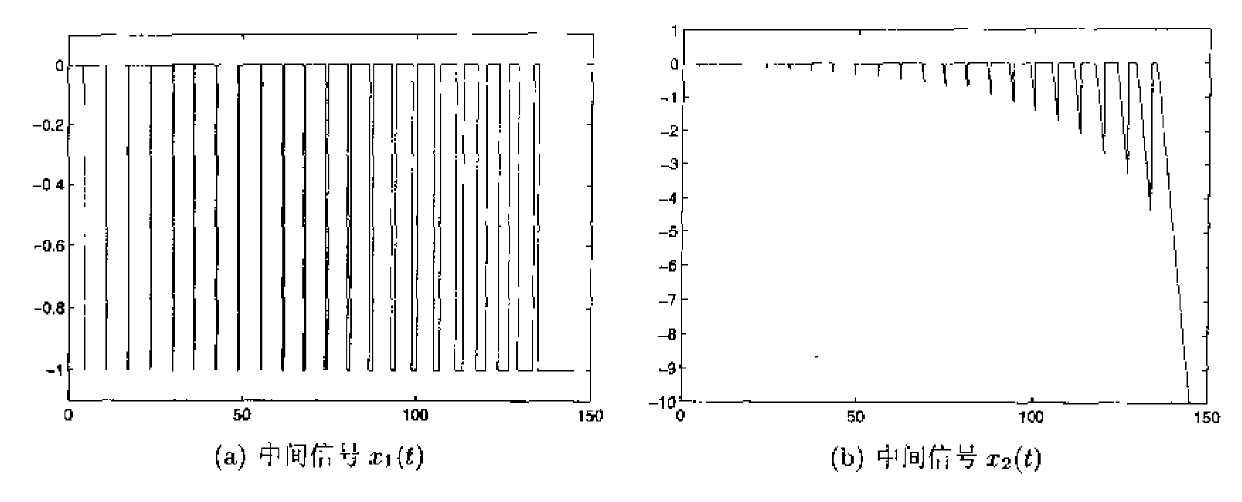
图 5-39 系统设置与输出结果

的时间越长，如果持续的时间大于 10，则后面的比较环节可以产生一个非 0 信号，驱动 STOP 模块，停止整个仿真过程。

这样就可以放心地设置一个大的仿真终止时间了，比如将其设置为 1000。启动仿真过程，则将得出如图 5-39 (b) 所示的输出结果。可见，该仿真过程在系统进入调节区域就自动停止了。

图 5-40 中给出了两个中间信号 $x_1(t)$ 和 $x_2(t)$ 的时间响应曲线，这些中间信号可以由下面的 MATLAB 语句直接绘制出来

```
>> plot(tout,yout(:,3)), set(gca,'ylim',[-1.1,0.1]);
   figure; plot(tout,yout(:,2)), set(gca,'ylim',[-10,1]);
```



(a) 中间信号 $x_1(t)$　　(b) 中间信号 $x_2(t)$

图 5-40　系统的中间信号仿真曲线

在得出的 $x_2(t)$ 信号中可以清晰地看出积分复位的作用。

从上面的例子可以看出，想判断一个信号持续长度并不是件容易的事，所以这里才用一个较麻烦的方法来搭建这个判定装置。好在这样的装置可以适用于一类这样任务的判定，从而利用停止仿真模块来根据系统实际情况设定仿真的终止时间。

5.2.2　输出信号的表盘与量计显示

前面介绍过，Simulink 提供了各种各样的曲线和数据输出模块，然而在实际的应用中经常需要以另外的形式显示结果，如实际过程控制等需要用各种各样仪表来显示信号。Global Majic 公司为 MATLAB/Simulink 提供了一系列基于 ActiveX 技术的表盘与量计显示部件。

ActiveX 部件是 Microsoft 公司提供的一种用于模块集成的新协议，它是 Visual Basic 工具箱的扩充部分。ActiveX 部件是一些遵循 ActiveX 规范编写的可执行代码，比如一个 `.exe`、`.dll` 或 `.ocx` 文件。在程序中加入 ActiveX 部件后，它将成为开发和运行环境的一部分，并为应用程序提供新的功能。ActiveX 部件保留了一些普通 VB 控件的属性、事件和方法。ActiveX 部件特有的方法和属性大大地增强了程序设计者的能力和灵活性。当前的新软件一般都支持 ActiveX 部件的嵌入，所以令现代的程序设计不再是一个个孤立的程序，而可以是一些在各个方面有优势的软件集成。这确实是软件业的重大革命。

可以通过 Simulink 直接进入 Dials & Gauges 模块集，或在 MATLAB 的命令窗口中键入命令 `dnglib`，则将打开一个如图 5-41 所示的界面，这就是表盘与量计模块集的主界面，从该界面可以访问该模块集的所有模块。

在该模块集中双击 Global Majic ActiveX Library (该公司的 ActiveX 库) 图标，则可以得出如图 5-42 所示的模块组。在该模块组中又包含了各种表盘与量计组，如双击其中的 Buttons & Switches (按钮与开关组) 则将得出如图 5-43 所示的内容。在该组中有各种各样的相关按钮，除了真正意义下的按钮和开关外，还有其他类似的模块，如锁和指示灯

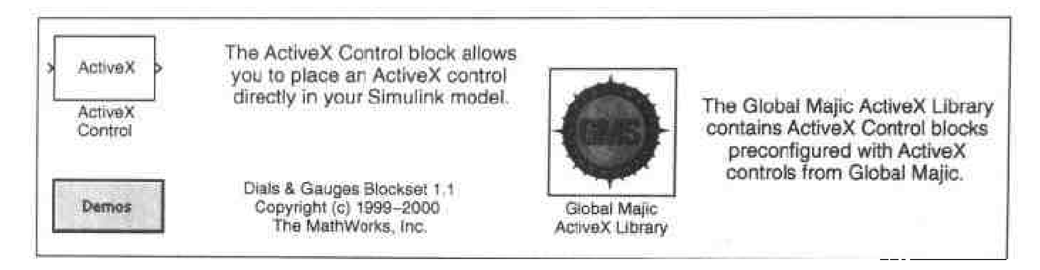


图 5-41 Dial & Gauges 模块集

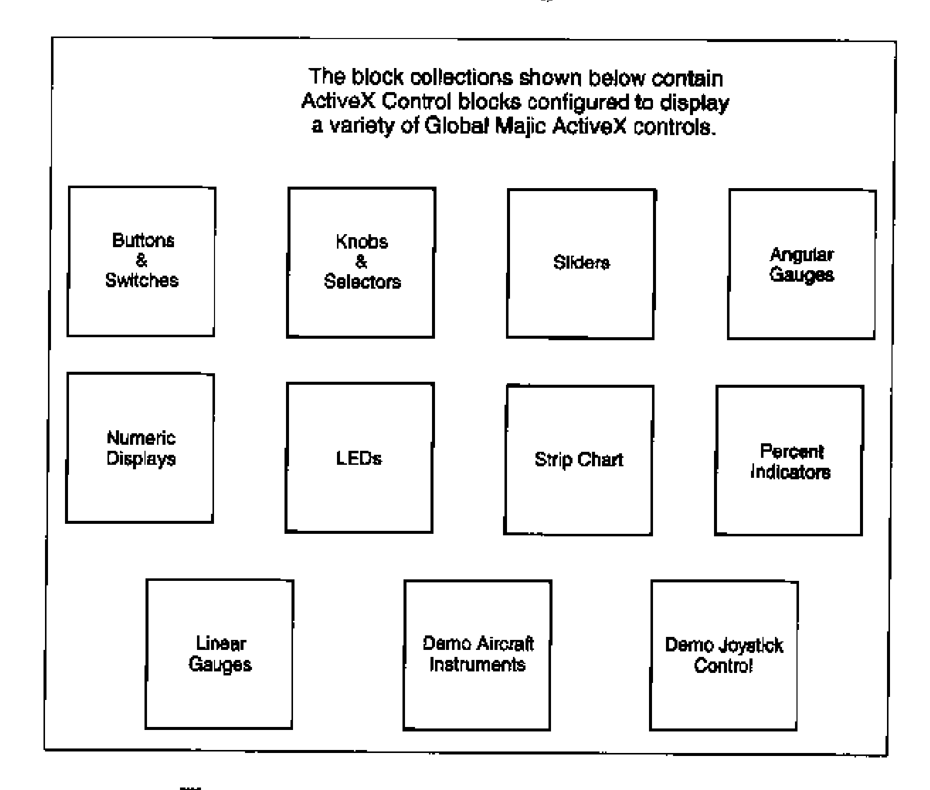


图 5-42 Global Majic 公司的 ActiveX 部件库

等。可以将其接到某个信号上，用更形象的方式来显示开关量。

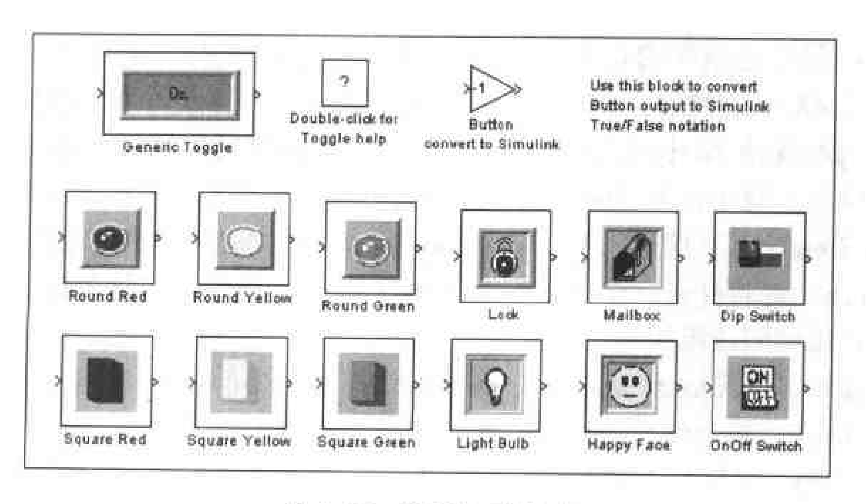


图 5-43 按钮与开关库

双击 Angular Gauges (圆形量计) 图标，则将打开一个如图 5-44 所示的量计库，该模块库包括了各种各样的仪表图标，例如安培表 (Amp Meter)、真空表 (Vacuum)、秒表 (Stop Watch) 和其他各类仪表。其实这里给出的各个仪表都可以由普通仪表 (Generic Angular Gauge) 变化而来，后面将通过例子来演示。

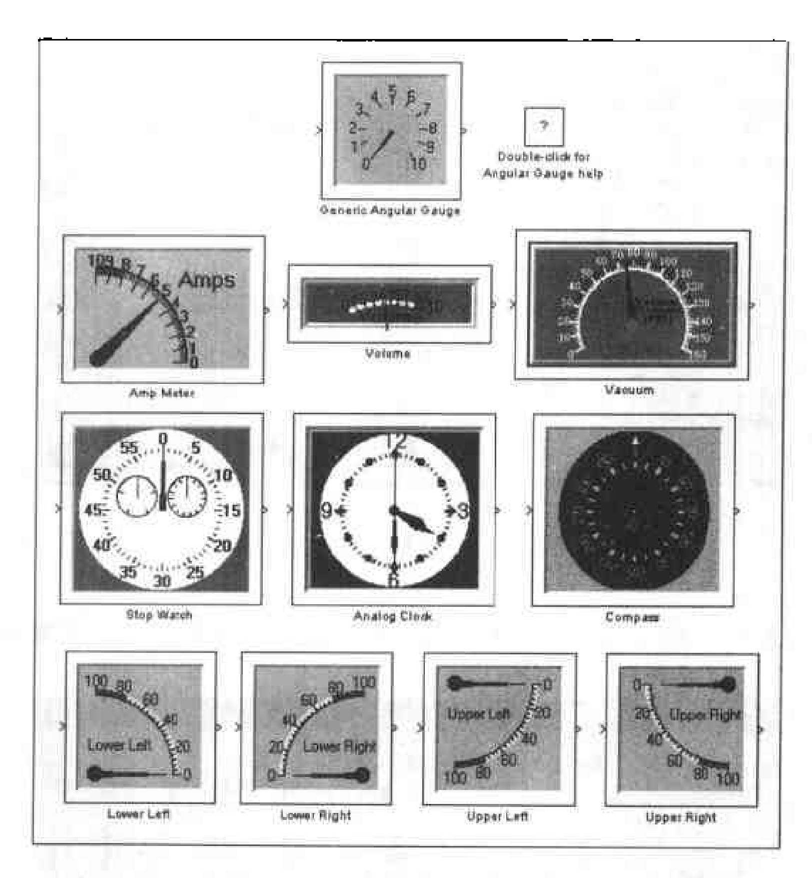


图 5-44 圆形表盘与量计库

〖例 5.14〗 考虑前面 PI 控制的例子 (例 5.12)，可以给输出端加一个真空表，如图 5-45 所示。在 Simulink 模型窗口中移动一个表盘模块和一个普通的 Simulink 模块的方法不完全一致。仔细观察表盘模块可以发现，其外框和内部图形之间有一个白色的边框，只有拖动这个区域才能真正地移动模块。

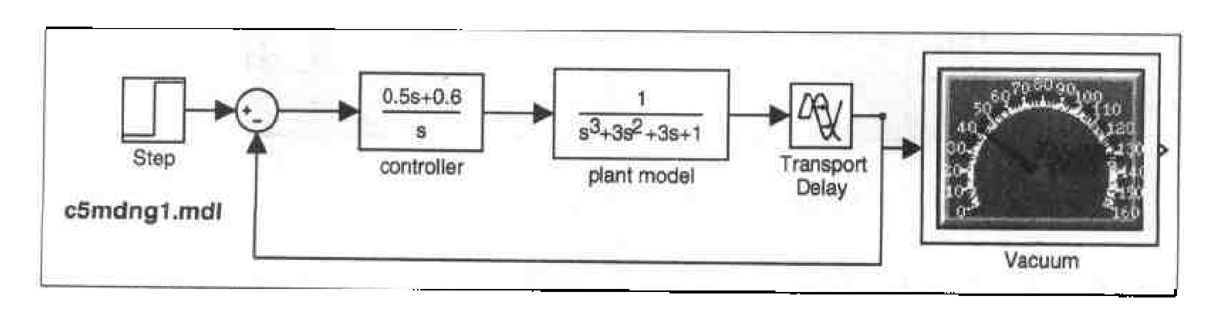


图 5-45 带有表盘输出的 PI 控制系统

从表盘上可以看出，预设的输出范围较大 (0~160)，而实际的输出信号范围不大于 2，这样使

得仿真结果的显示可读性不是很理想。可以双击该模块将得出如图 5-46 所示的对话框，再选择其

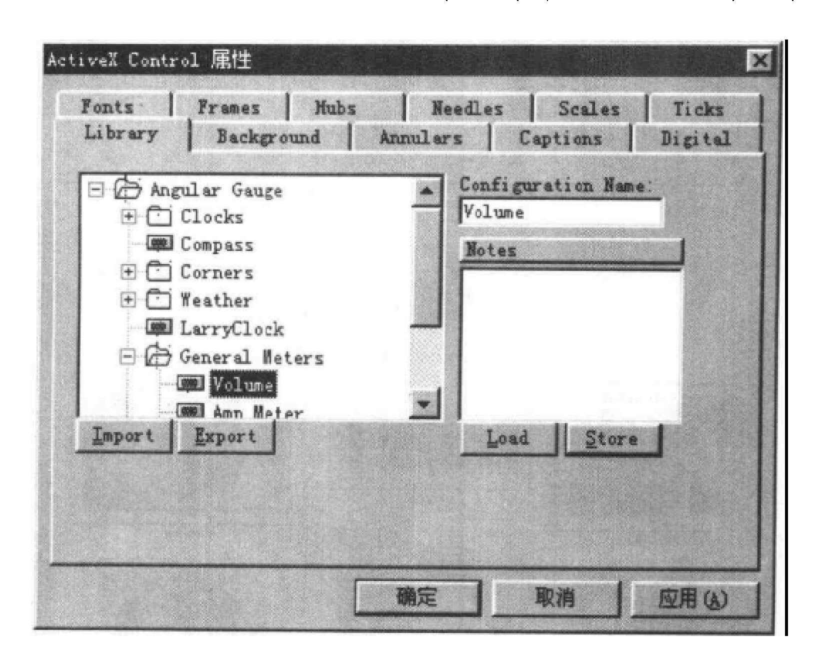


图 5-46　表盘属性设置对话框

中的 Scales (标度) 标签，则可以得出如图 5-47 所示的对话框，在该对话框中可以将标度的范围设

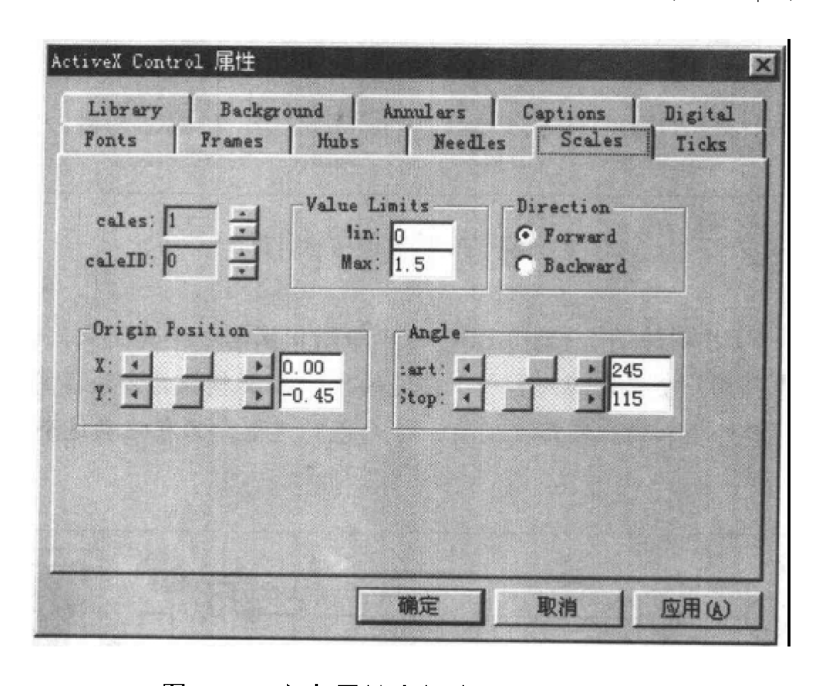


图 5-47　表盘属性中标度的设置对话框

置为 0 到 1.5，再打开 Ticks (刻度) 标签，将右下角的 DeltaValue (变化值) 填写为 0.1，则将得出如图 5-48 所示的框图形式，其中表盘按新指定的值发生了变化。完成了设置，就可以开始数值仿真了。其实仿真的过程是特别快的，几乎无法仔细地观察指针的变化，除非人为地将仿真步长设置成一个非常小的值。

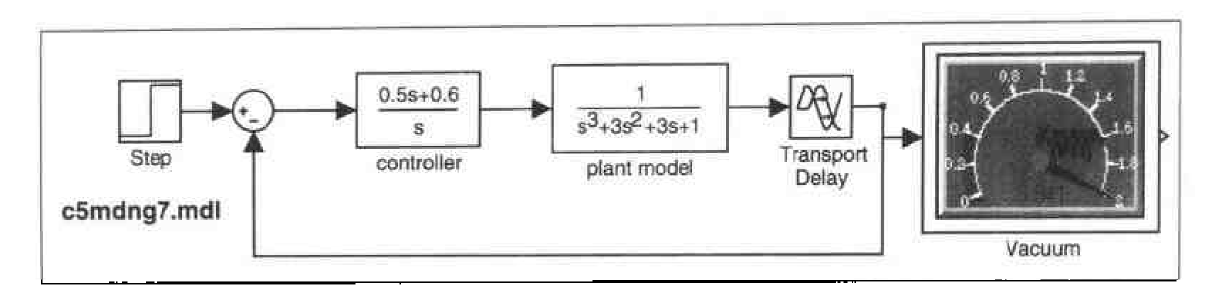


图 5-48 表盘标度参数修改后的系统模型

还可以利用该模块的属性设置对话框进行各种其他的设置，这些设置很多都是很自然的，例如，返回图 5-46 所示的对话框，可以进入 Library (模块库) 标签，双击表盘类型中的 Volume (流量) 选项，则将得出如图 5-49 所示的系统框图表示，注意这时整个标度也将发生变化，所以用户可以重复上述的过程重新修正标度设置。

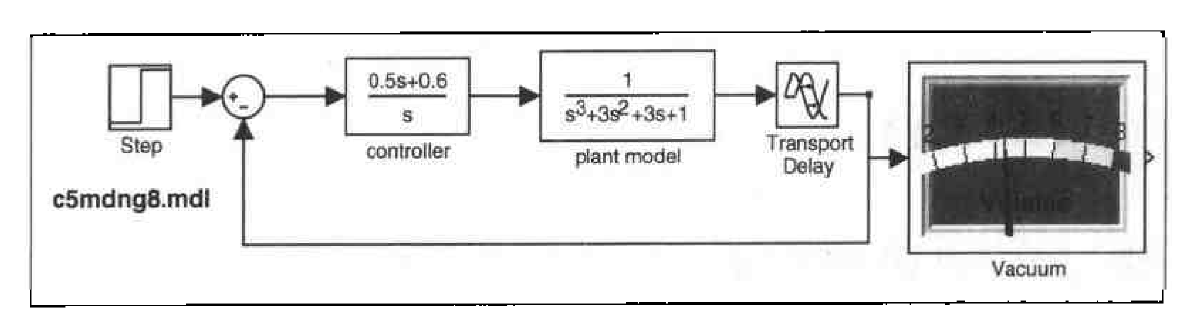


图 5-49 表盘类型修改后的系统模型

5.2.3 输出的数字信号处理

Simulink 在 Simulink Extras (其他模块) 组中的 Additional Sinks (附加输出池) 提供了一些数字信号处理的模块，如图 5-50 所示。另外，在 DSP Blockset (数字信号处理模块

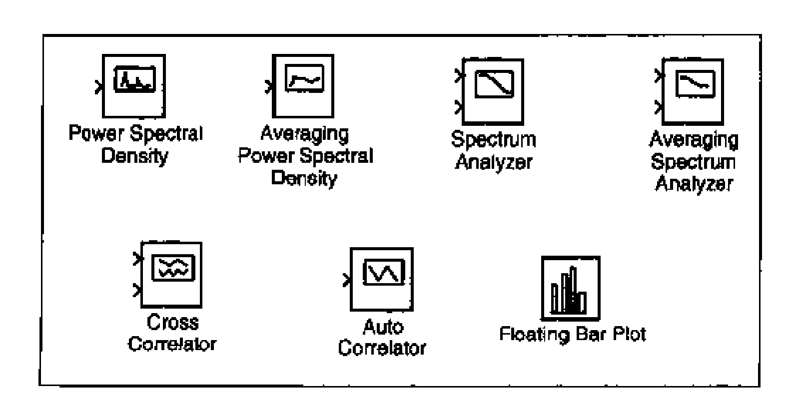


图 5-50 Simulink 附加输出池库

集) 中提供了更强大的模块库，如图 5-51 所示。在本节中将通过例子来演示数字信号处理的功能。

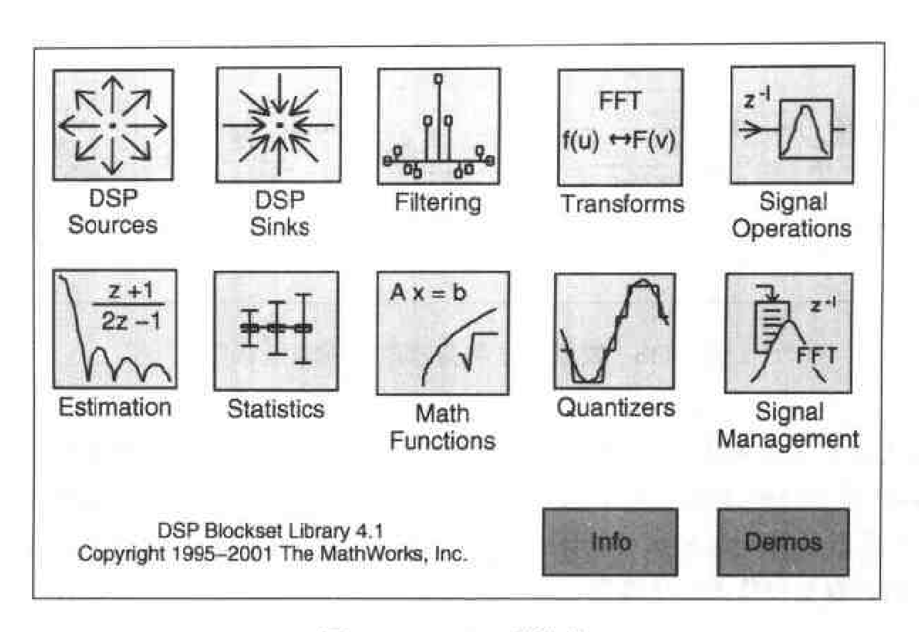


图 5-51 DSP 模块集

【例 5.15】 假设输入信号为

$$x(t) = \sin(t) + 1.2\cos(2t)$$

可以按照图 5-52 (a) 所示的方式构造出 Simulink 仿真框图，在该框图中，使用了两个正弦输入模块，按照给出函数中的参数分别设置这两个模块的参数。例如，在第 2 个正弦输入模块的参数对话框中可以按图 5-52 (b) 的形式给出数据，亦即其幅值为 1.2，频率为 2rad/sec，初始相位为 $\pi/2$，这样 $1.2\sin(2t+\pi/2) = 1.2\cos(2t)$ 即为要求的信号。这两个模块叠加之后，将其结果连接到自相关分析器的模块上。

(a) 仿真框图

(b) 正弦输入模块参数

(c) 自相关函数显示

图 5-52 输入信号的自相关分析

如果想使用自相关模块，则需要将系统仿真设置为定步长的算法，且可以选择步长为 0.1，这样启动仿真过程，则将得出如图 5-52 (c) 所示的结果，其中上图为时间响应曲线，下图为其自相关函数。

如果将图 5-52 (a) 所示模型中的 Auto Correlator 模块用 Power spectral density 模块取代，就能立即得出如图 5-53 所示的功率谱密度。

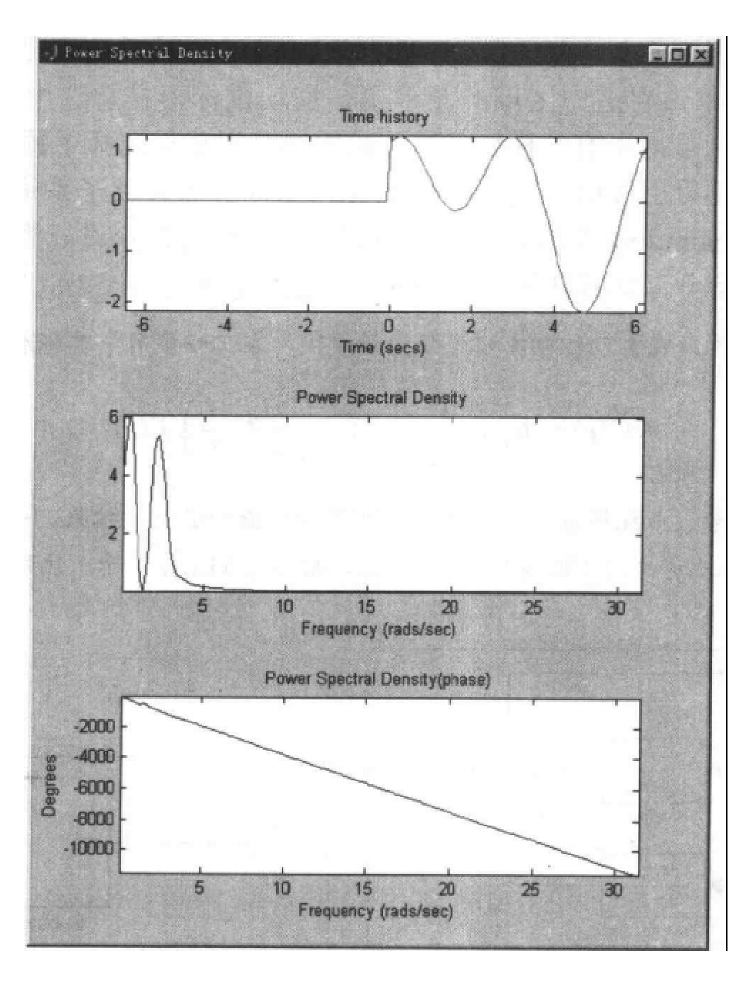


图 5-53 信号的功率谱密度

5.3 子系统与模块封装技术

在系统建模与仿真中，经常遇到很复杂的系统结构，难以用一个单一的模型框图进行描述。通常地，需要将这样的框图分解成若干个具有独立功能的子系统，在 Simulink 下支持这样的子系统结构。另外用户还可以将一些常用的子系统封装成为一些模块，这些模块的用法也类似于标准的 Simulink 模块，更进一步地，还可以将自己开发的一系列模块做成自己的模块组或模块集。本节中，将系统地介绍子系统的构造及应用、模块封装技术和模块库的设计方法，并通过一个较复杂系统的例子来演示子系统的构造和整个系统的建模。

5.3.1 子系统的处理

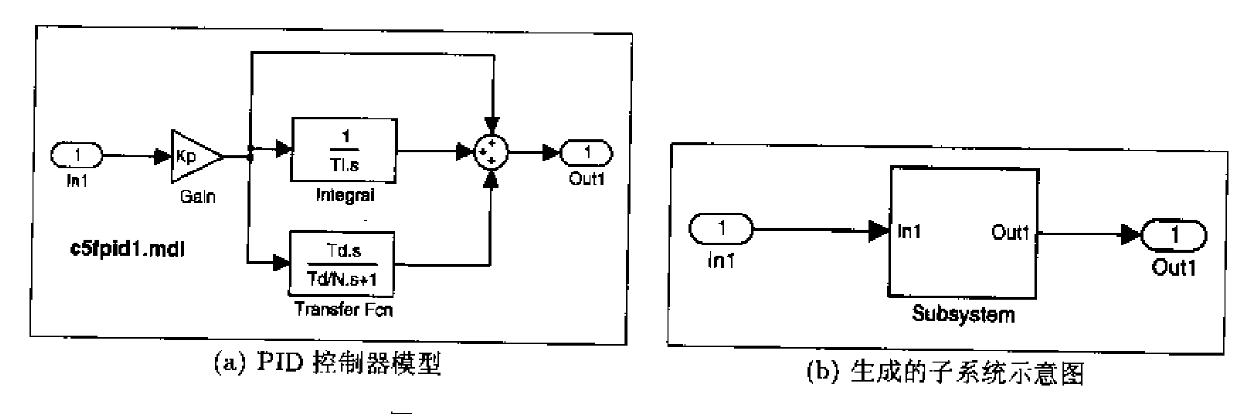
要建立子系统，首先需要给子系统设置输入和输出端。子系统的输入端由 Sources 模块组中的 In 来表示，而输出端用 Sinks 模块组的 Out 来表示。注意，如果使用早期的 Simulink 版本，则输入和输出端子应该在 Signals & Systems 模块组中给出。在输入端和输出端之间，用户可以任意地设计模块的内部结构。

当然，如果已经建立起一个方框图，则可以将想建立子系统的部分选中，具体的方法是用鼠标器左键单击要选中区域的左下角，拖动鼠标器在想选中区域的右上角处释放，则可以选中该区域内所有的模块及其连接关系。用鼠标选择了预期的子系统构成模块与结构之后，则可以用 Edit | Create Subsystem 菜单项来建立子系统。如果没有指定输入和输出端口，则 Simulink 会自动将流入选择区域的信号依次设置为输入信号，将流出的信号设置成输出信号，从而自动建立起输入与输出端口。

【例 5.16】 PID 控制器是在自动控制中经常使用的模块，在工程应用中其标准的数学模型为

$$U(s) = K_p\left(1+\frac{1}{T_i s}+\frac{sT_d}{1+sT_d/N}\right)E(s) \tag{5.3}$$

其中采用了一阶环节来近似纯微分动作，为保证有良好的微分近似的效果，一般选 $N \geq 10$。可以由 Simulink 环境容易地建立起 PID 控制器的模型，如图 5-54 (a) 所示。注意，这里的模型含有 4



(a) PID 控制器模型 (b) 生成的子系统示意图

图 5-54 PID 控制器的 Simulink 描述

个变量，Kp, Ti, Td 和 N，这些变量应该在 MATLAB 工作空间中赋值。

绘制了原系统的框图，可以选中其中所有的模块，例如可以使用 Edit | Select All 菜单项来选择所有模块，也可以用鼠标拖动的方法选中。这样就可以用菜单项 Edit | Create Subsystem 来构造子系统了，得出的子系统框图如图 5-54 (b) 所示。双击子系统图标则可以打开原来的子系统内部结构窗口，如图 5-54 (a) 所示。

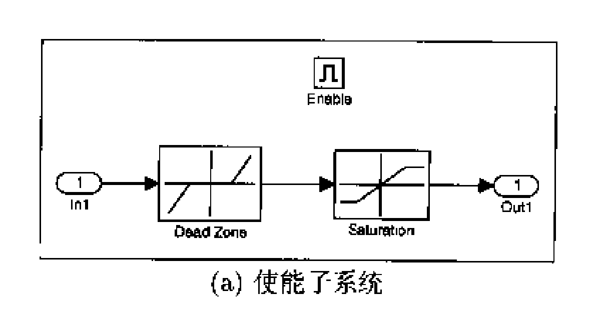
5.3.2 条件执行子系统

在 Simulink 中，允许某个子系统在给定的控制信号下执行，这样的子系统称为条件执行子系统 (conditionally executed subsystems)，当前版本共支持下面三种控制结构：

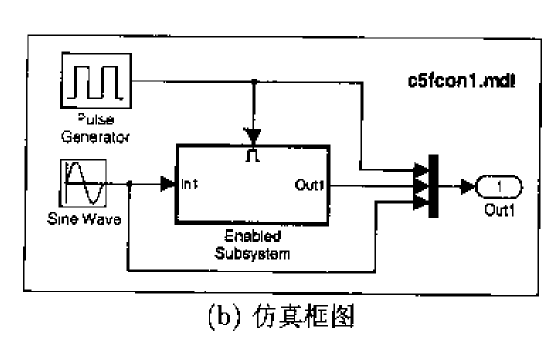
- **使能子系统** (enabled subsystetm) 将子模块条件信号称为控制信号，控制信号分成“允许” (enable) 和“禁止”(disabled) 两种，在允许信号控制下 (Simulink 的约定下，当控制信号为正时将模块设置为允许状态，否则为禁止状态)，可以执行子系统中的模块，否则将禁止其功能。为保证整个系统的连贯性，在禁止状态下子系统仍然有输出信号，用户可以选择继续保持禁止前的信号，或复位子系统，强制使其输出零信号。
- **触发子系统** (triggered subsystem) 在控制信号满足某种变化要求的瞬间可以触发 (激活) 子系统，然后保持子系统输出的状态，等待下一个触发信号。它允许用户自己设置在控制信号的上升沿、下降沿或控制信号变化时触发子系统。
- **使能触发子系统** (enabled and triggered subsystetm) 在使能状态下被触发时将激活该子系统，否则将禁止子系统。

下面将通过例子来演示条件执行子系统的构造和功能，并介绍有关上升沿和下降沿触发的概念。

〖例 5.17〗 考虑由死区和饱和非线性环节串联起来的结构，可以将按如下的方式建立起一个使能子系统：首先打开一个空白的模型编辑窗口，将 Subsystems 模块组中的 Enabled subsystem 模块拖动到模块组中。双击该模块，则将得到子系统编辑窗口，在该窗口中将所需的非线性模块建立起来，并设置死区模块填写的参数为 $(-1,1)$，饱和模块填写的是 $(-3,3)$，则可以建立起如图 5-55 (a) 所示的子系统。



(a) 使能子系统



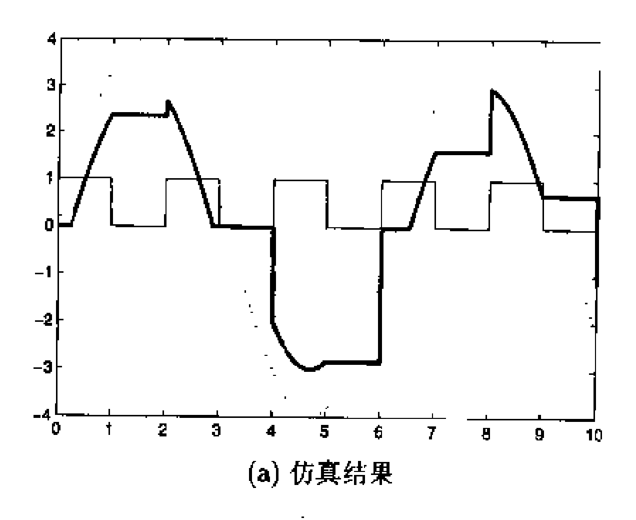
(b) 仿真框图

图 5-55 使能子系统框图

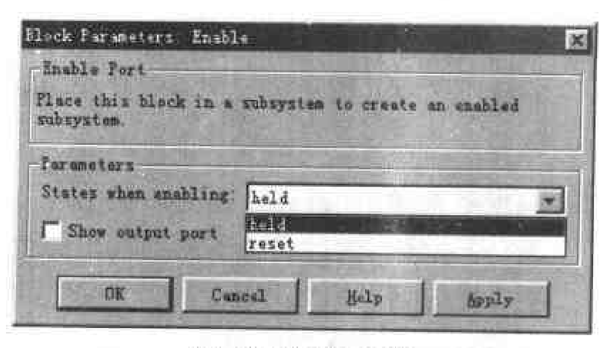
在 MATLAB R12 (即 6.0 版) 和早期版本下构造使能子系统的方式略有不同，因为它没有单独提供子系统模块组。在该版本中，需要将普通的子系统先建立起来，然后将信号与系统 (Signals & Systems) 模块组中的使能 (Enable) 模块、触发 (Trigger) 模块移动到该组中，方能建立起相应的带有使能、触发功能的子系统。

回到原来的模型窗口，给子系统加一个幅值为 4 的正弦输入信号，并给使能控制信号加一个脉冲信号，可以得出如图 5-55 (b) 所示的仿真框图。选择定步长仿真，并将仿真步长设置为 0.02，则可以得出如图 5-56 (a) 所示的仿真结果。为了更好地观测输出信号，加粗了该曲线，从几条曲线的比较中可以看出使能信号的作用。

进入子系统模型，双击使能图标，则将得出如图 5-56 (b) 所示的对话框，在该对话框中可以



(a) 仿真结果

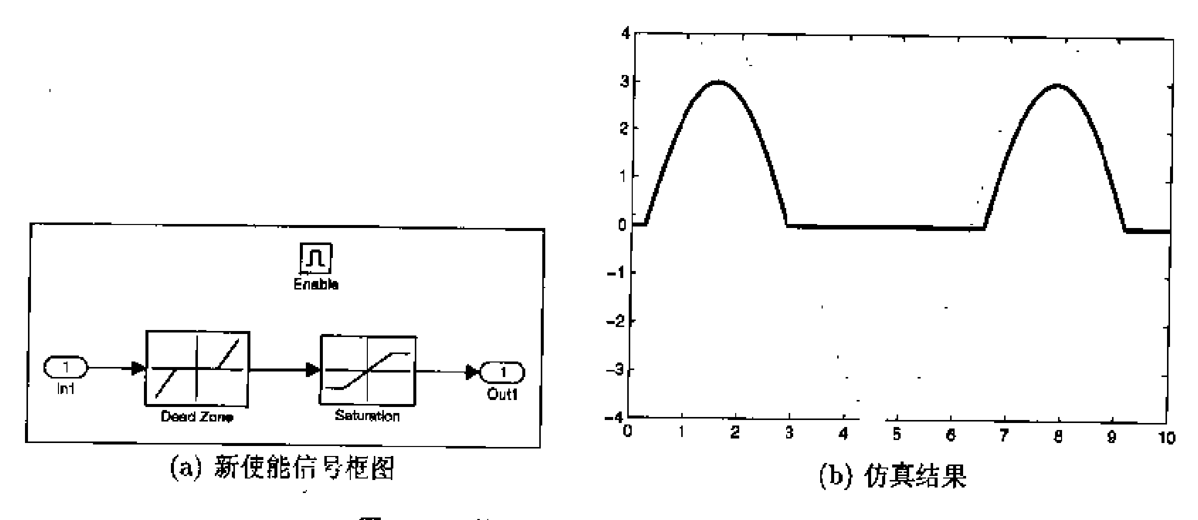


(b) 使能信号对话框

图 5-56　使能信号仿真结果与设置

选择使能开始时状态的值 (States when enabling)，其值可以选择 reset (复位) 和 held (保持当前状态)，不过在这个静态非线性系统中二者没有区别。

按图 5-57 (a) 中给出的方式改变使能信号，则将得出如图 5-57 (b) 所示的仿真效果。



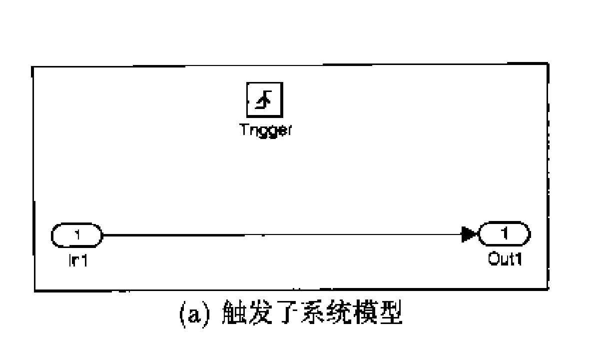
(a) 新使能信号框图　　(b) 仿真结果

图 5-57　修改使能信号后的框图与结果

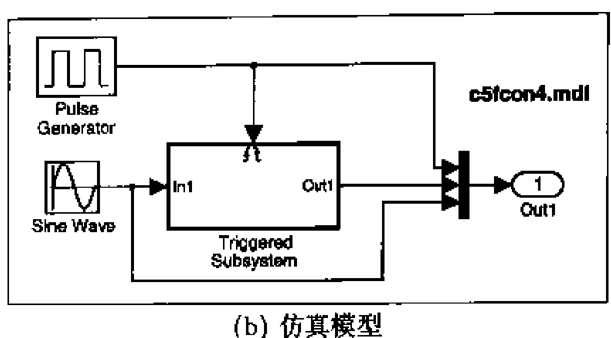
【例 5.18】 在这里将演示触发子系统的性质，假设重新改写子系统的内部结构，让其输入端直接连接到其输出端，这样就可以将一个触发子系统模块复制到模型编辑窗口。双击该子系统图标，则将得出如图 5-58 (a) 所示的子系统模型，这样就可以在模型编辑窗口中构造主系统模型，如图 5-58 (b) 所示。

双击子系统中的触发器图标，则将得出如图 5-59 (a) 所示的对话框，可以看出，Trigger type 栏目可以选择触发器的类型，如上升沿触发 (rising)、下降沿触发 (falling) 和上升沿和下降沿触发 (either)，还可以选择用回调函数 (function-call) 等。

启动仿真过程，在仿真结束后给出 `plot(tout,yout)` 命令，将得出如图 5-59 (b) 所示的曲线

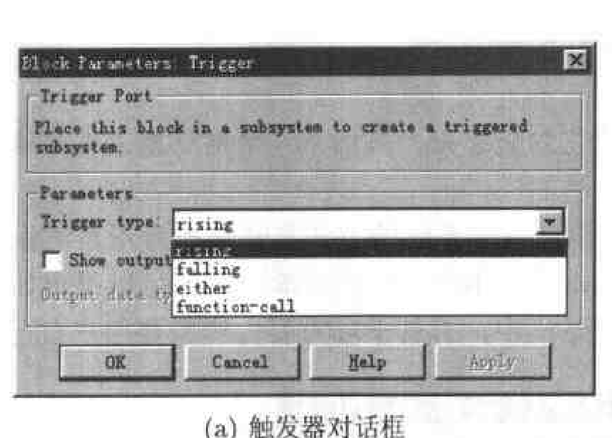


(a) 触发子系统模型

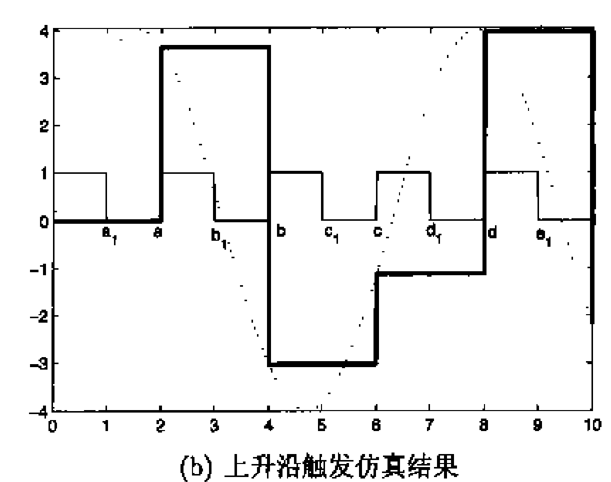


(b) 仿真模型

图 5-58 触发子系统的模型




(a) 触发器对话框



(b) 上升沿触发仿真结果

图 5-59 触发器设置与仿真

图，为方便起见，这里仍用粗线表示系统的输出信号；另外，用符号 a, a_1, b, b_1 等表示控制信号变化点，亦即带有下标的表示下降沿，不带下标的表示上升沿。可以看出，在上升沿处瞬时输出信号直接取输入信号的值，且在触发过程完成时保持该值，等待下一个触发信号。

如果在图中给出的对话框内改变触发类型为下降沿，则将得出如图 5-60 (a) 所示的仿真结果，如果选择了 either 触发类型，则将得出如图 5-60 (b) 所示的仿真结果。

Simulink 还允许将使能端子和触发端子共同控制子系统，在系统处于使能状态下，触发事件发生则将激活子系统，如果系统处于非使能状态，则将忽略触发信号的作用。

Simulink 中还提供了更复杂的流程控制，例如转移、开关和循环等控制结构，这些结构是依赖 Stateflow 构造的，后面介绍 Stateflow 的章节中将详细介绍。

5.3.3 模块封装技术

所谓封装 (masking) 模块，就是将其对应的子系统内部结构隐含起来，以便访问该模块时只出现一个参数设置对话框，将模块中所需要的参数用这个对话框来输入。其实 Simulink 中大多数的模块都是由更底层的模块封装起来的，例如传递函数模块，其内部结构是不可见的，它只允许双击打开一个参数输入对话框来读入传递函数的分子和分母

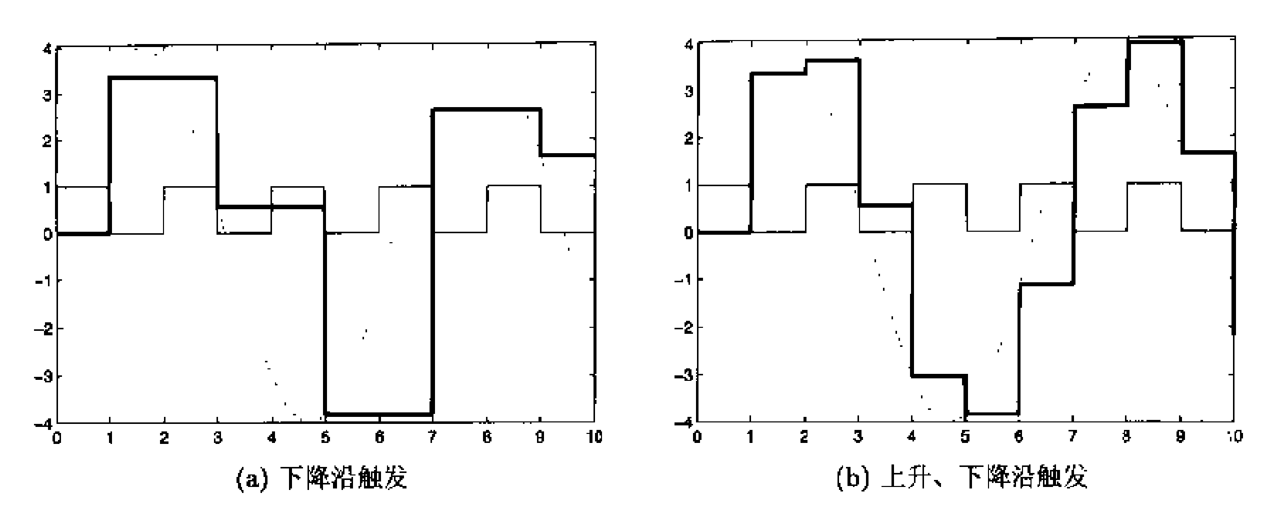


图 5-60 其他触发类型的仿真结果

参数。在前面介绍的 PID 控制器中，也可以给它封装起来，只留下一个对话框来输入该模块的 4 个参数。

如果想封装一个用户自建模型，首先应该用建立子系统的方式将其转换为子系统模块，选中该子系统模块的图标，再选择 Edit | Mask Subsystem 子菜单项，则可以得出如图 5-61 所示的模块封装编辑程序界面，在该对话框中，有若干项重要内容需要用户自己

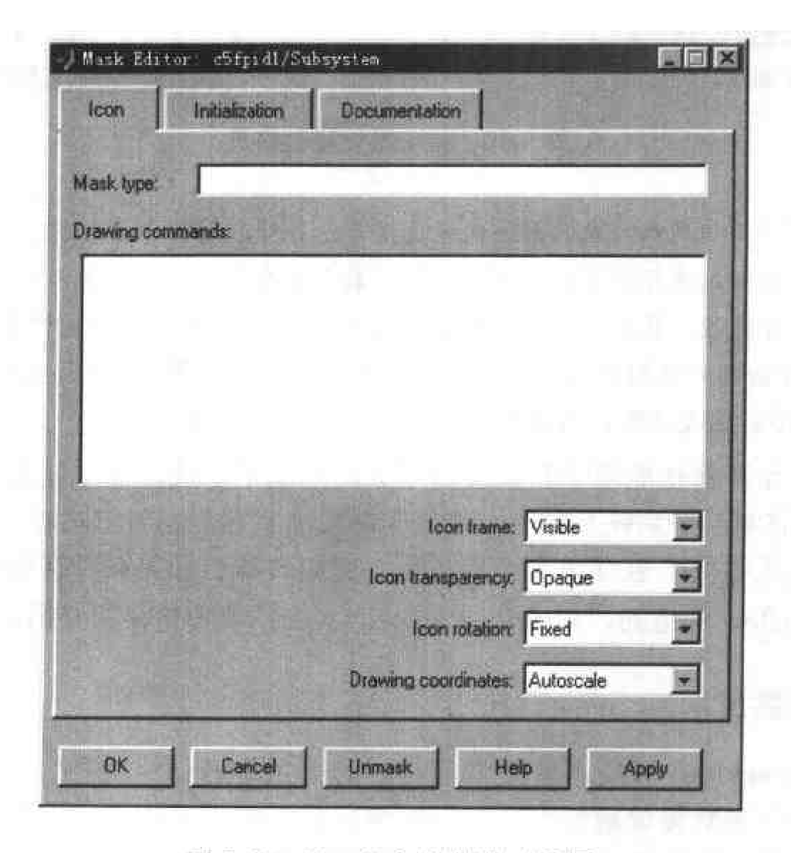


图 5-61 Simulink 的封装对话框

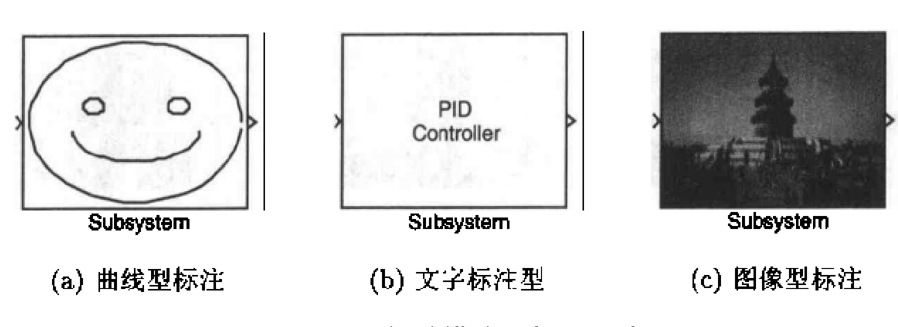
填写，例如：

- Mask type (**封装种类**) 编辑框的内容可以任意填写，在其中填写一个您认为合适的字符串即可，这些编辑框都是可以接受中文字符的。
- Drawing commands (**绘图命令**) 编辑框允许给该模块图标上绘制图形，例如可以使用 MATLAB 的 `plot()` 函数画出线状的图形，也可以使用 `disp()` 函数在图标上写字符串名，还允许用 `image()` 函数来绘制图像。

如果想在图标上画出一个“笑脸”，则可以采用下面的 MATLAB 命令，分别绘制出四条曲线，其中外部画一个单位圆表示“脸”，两个小圆，半径为 0.1，圆心分别在 (-0.4,0.2) 和 (0.4,0.2) 处绘制两个圆，表示眼睛，在底部画一个半椭圆，表示嘴，这样就可以将下面的命令填写到此编辑框中

```
plot(cos(0:.1:2*pi),sin(0:.1:2*pi),-0.4+0.1*cos(0:0.1:2*pi),...
   0.2+0.1*sin(0:0.1:2*pi),...
   0.4+0.1*cos(0:0.1:2*pi),0.2+0.1*sin(0:0.1:2*pi),...
   0.6*cos(0:.1:pi),-0.1-0.4*sin(0:.1:pi))
```

从而可以在图标上标注成如图 5-62 (a) 所示的形式。



(a) 曲线型标注　(b) 文字标注型　(c) 图像型标注

图 5-62　封装模块的标注形式

从上面给出的绘图语句可以看出，原本很简单的绘图问题似乎复杂化了，例如在各个圆的绘制中都使用了 `0:.1:2*pi` 变量，而在 MATLAB 绘图中显然可以用一个变量来取代它，所以可以在该编辑框中试着填写

```
t=0:.1:2*pi; plot(cos(t),sin(t));
line(-0.4+0.1*cos(t),0.2+0.1*sin(t));
line(0.4+0.1*cos(t),0.2+0.1*sin(t));
t=0:.1:pi; line(0.6*cos(t),-0.2-0.4*sin(t));
```

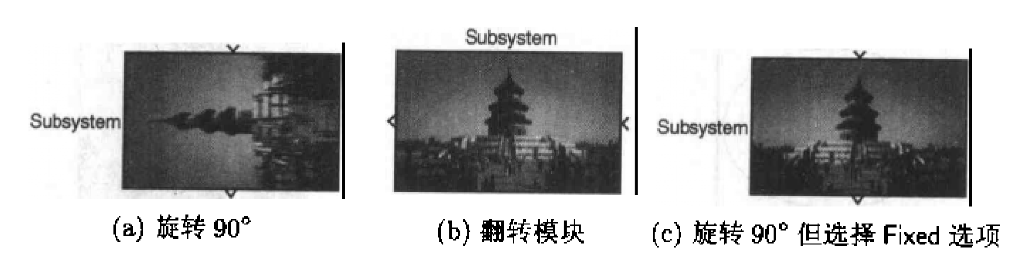
显然这些语句在 MATLAB 的图形窗口中会直接绘制出“笑脸”，然而这些语句若放在该编辑框中将得出错误信息 “Warning: Unrecognized function encountered in mask display command.”，说明变量 `t` 在编辑框内不能传递，所以需要将该变量隐含在绘图命令中，而不能单独使用。

还可以使用 `disp('PID\nController')` 语句对该图标进行文字标注，这将得出如图 5-62 (b) 所示的图标显示，其中 `\n` 表示换行。

在该编辑框中给出 image(imread('tiantan.jpg')) 命令将一个图像文件在图标上显示出来，如图 5-62 (c) 所示。

在即将正式推出的 Simulink 5.0 版本中，允许在字符串中使用 TEX 命令，这样就可以在模块上或输入输出端口上写出复杂的字符串描述，如添加数学公式。

- Icon frame (**图标边框**) 选项可以为 Visible (可见的) 和 Invisible (隐含的)，其中前者为默认状态，大多数 Simulink 模块均带有可见的边框。
- Icon transparency (**图标透明与否**) 属性有两种选择，Opaque (不透明的，为默认属性) 和 Transparent (透明的)。如果采用默认的选项，则模块端口的信息将被图标上的图形完全覆盖，所以如果想显示端口名称，则应该采用 Transparent 选项。
- Icon rotation (**图标是否旋转**) 属性有两种选择，Fixed (固定的，默认选项) 和 Rotates (旋转)，后者在旋转或翻转模块时，也将旋转该模块的图标，例如若选择了 Rotates 选项，则将得出如图 5-63 (a) 和 (b) 所示的效果。从旋转效果看，似乎翻转的模块其图标没有变化，仔细观察该图标可以发现，其图标为原来图标的左右翻转。若选择了 Fixed 选项，则在模块翻转时不翻转图像，如图 5-63 (c) 所示。



(a) 旋转 90°　(b) 翻转模块　(c) 旋转 90° 但选择 Fixed 选项

图 5-63　图标的旋转和翻转

- Drawing coordinates (**绘图坐标系**) 属性有三种选项，Pixels (像素点)、Autoscale (自动定标，默认选项) 和 Normalized (归一化的)。

封装模块的另一个关键的步骤是建立起封装的模块内部变量和封装对话框之间的联系，选择封装编辑程序的 Initialization 标签，则将得出如图 5-64 所示的形式，其中间的区域可以编辑变量与对话框之间的联系。

可以按下 Add 按钮和 Delete 按钮来指定和删除变量名，例如在前面的 PID 控制器的例子中，可以连续按下 4 次 Add 按钮，为该控制器的 4 个变量准备位置。单击第一个参数位置，得出如图 5-65 (a) 所示的显示，可以在 Prompt (提示) 栏目中填写该变量的提示信息，如 Proportional Kp，然后在 Variable (变量) 栏目中填写出想关联的变量名 Kp，注意该变量名必须和框图中的完全一致。

还可以采用相应的方式编辑其他变量的关联关系。在编辑栏中最后的 Control type (控件类型) 栏目的默认值为 Edit，表示用编辑框来接受数据。如果想让滤波常数 N 只取几个允许的值，则可以将该控件选择为 Popup (列表框) 形式，并在 Popup string (列表字符串) 栏目上填写 10 | 100 | 1000，如图 5-65 (b) 所示。

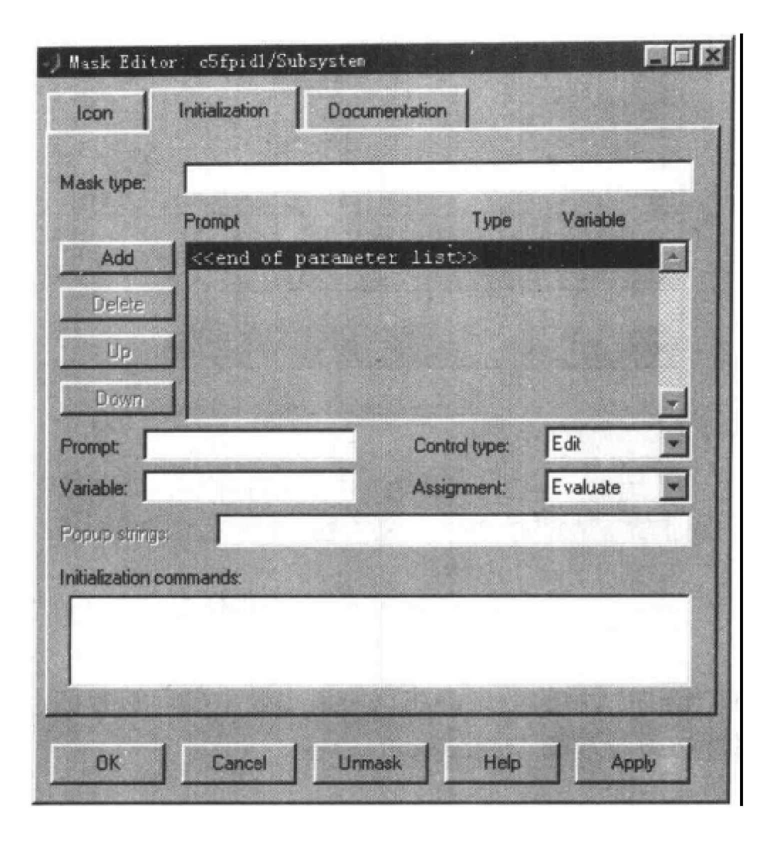
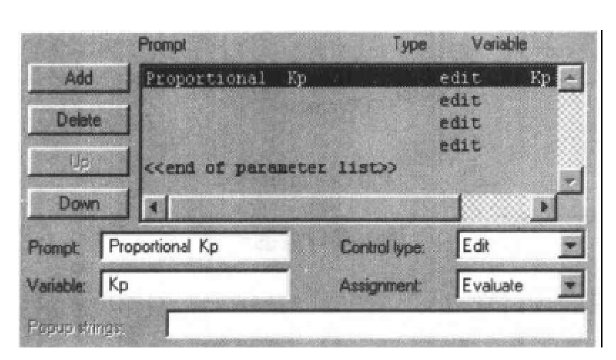
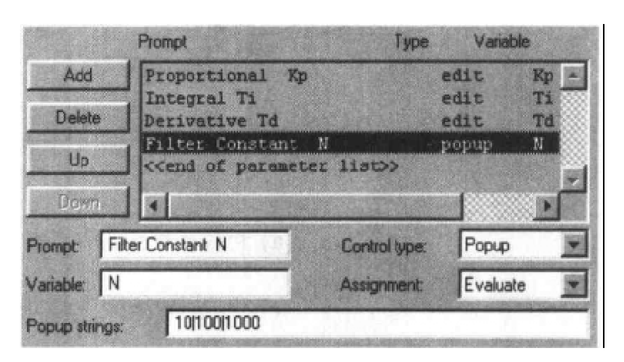


图 5-64 封装后模块的参数输入对话框



(a) K_p 变量编辑



(b) 列表框型变量编辑

图 5-65 封装变量的关联设置

每个变量的位置还可以调整，可以使用 Up 和 Down 按钮来修改次序。用户还可以进一步选择 Documentation 标签来为此模块编写说明文字，这样一个子系统的封装就完成了，模块封装完成，就可以在其他系统里直接使用该模块了，双击封装模块，则可以得出如图 5-66 所示的对话框，允许用户输入 PID 控制器的参数。注意，这里的滤波常数 N 由列表框给出，允许的取值为 10, 100 或 1000。

在封装的模块上右击鼠标键，可以打开快捷菜单，其中的 Look under mask 菜单项允许用户打开封装的模块，如图 5-67 (a) 所示，用户可以修改其中的输入和输出端口的名字，例如将输入的端口修改成 `error`，将输出的端口修改为 `control`，则修改后的封装模

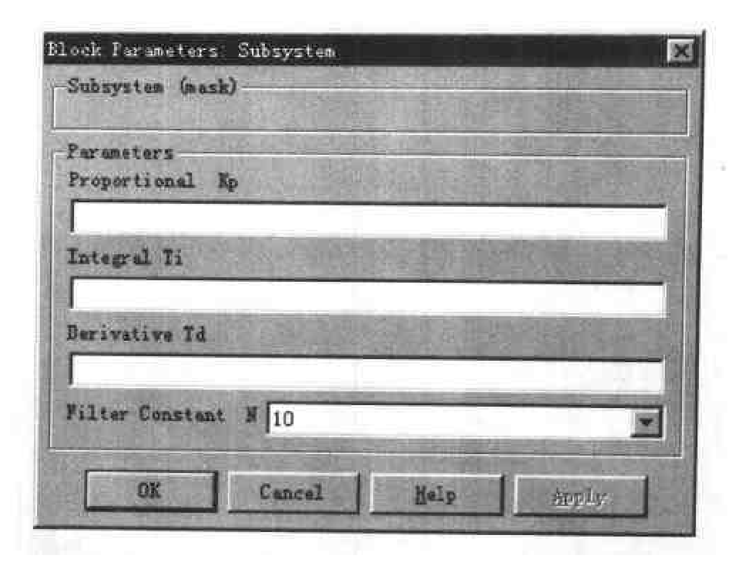
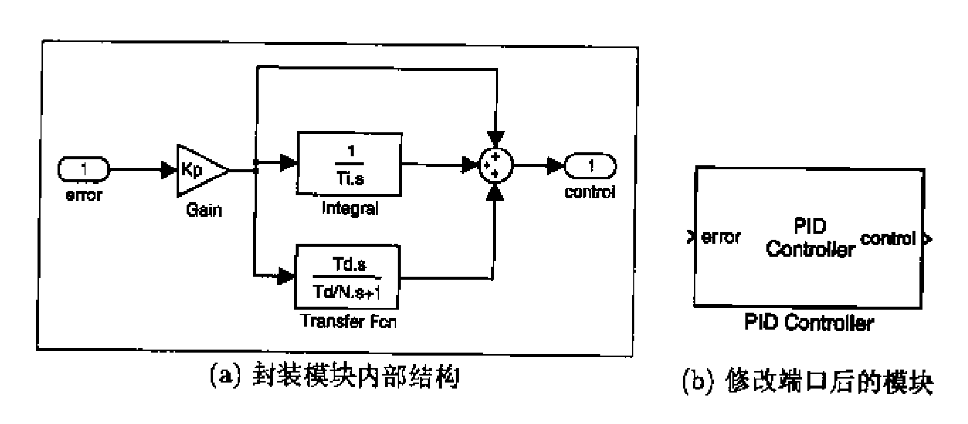


图 5-66　封装模块调用对话框

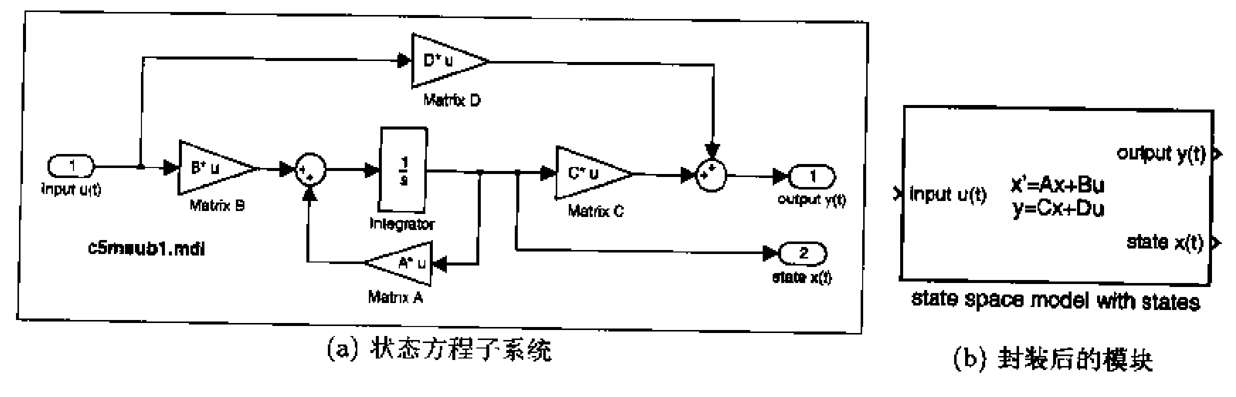
块会自动变为图 5-67 (b) 中所示的效果，注意如果想显示端口的名称，则封装对话框中的 Icon transparency 属性必须设置成 Transparent。



(a) 封装模块内部结构　　(b) 修改端口后的模块

图 5-67　封装变量的端口修改

〖例 5.19〗 再考虑例 5.2 中介绍的状态方程模型，可以建立起如图 5-68 (a) 所示的子系统，并

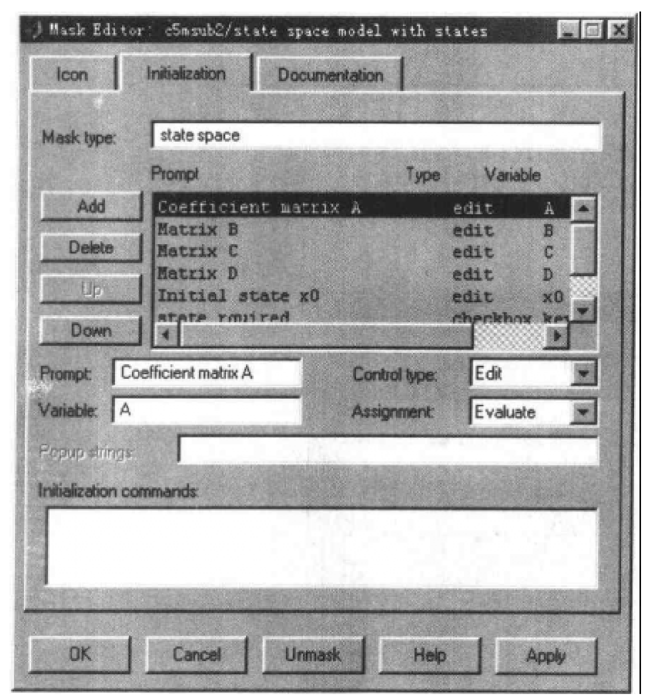


(a) 状态方程子系统　　(b) 封装后的模块

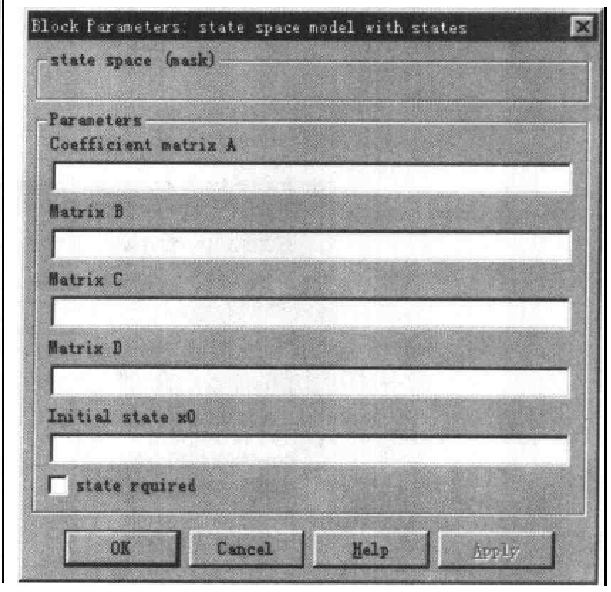
图 5-68　状态方程模型

将该子系统封装成如图 5-68 (b) 所示的形式。其中封装模块时，Drawing commands 栏目填写为 disp('x''=Ax+Bu\ny=Cx+Du')，而 Icon transparency 栏目设置成 Transparent。

在建立参数关联关系时，可以按照图 5-69 (a) 所示的填写参数关系，其中前几个参数



(a) Initialization 对话框



(b) 参数输入对话框

图 5-69 状态方程模型的封装

$(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{D})$ 为状态方程的矩阵，可以使用一般方式设置即可。期望系统的初始状态 x0 在不给入任何输入时取默认值；另外，设置了一个端口控制变量 States，为复选框 checkbox 型变量，目的是在其未被选中时隐含状态变量输出端口，这里将不演示如何隐含模块的输出端口，只在习题中给出用 S-函数实现这样的功能的要求。

5.3.4 组建自己的模块库

封装的模块可以用 Edit | Edit mask 菜单项来重新编辑，该菜单将打开如图 5-61 所示的对话框，重复前述的步骤就可以修改封装的参数了。要修改模块内部的结构，则应该右击该模块，打开快捷菜单，从中选择 Look under mask 菜单项，这样将打开构成该模块的子系统结构图，可以修改其实际框图。

如果用户自己建立了很多封装的模块，则往往需要再建立一个模块库来存储这些模块。另外，用户也可以将常用的一组模块建立一个单独的模块库，以便自己调用。

创建模块库的方法是：选中 Simulink 浏览器中的 New | Library 菜单项，这样将打开一个空白的模块库窗口，如图 5-70 所示。可以将该模块库存为新的文件，如 my_blks。可以选择 File | Model properties 菜单项来设置该模块库的属性，这样将打开一个如图 5-71 所示的对话框，在该对话框中还可以输入相关的内容。

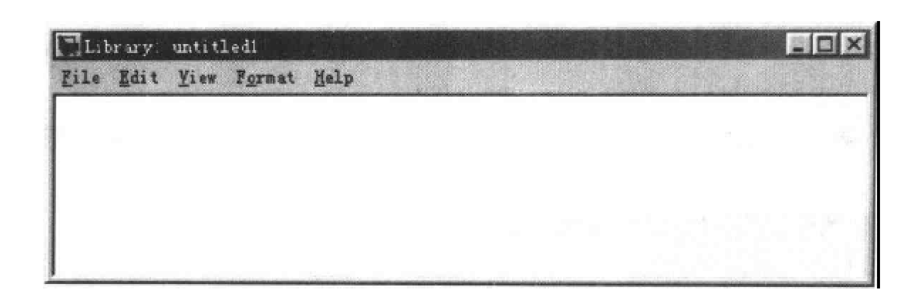


图 5-70 空白模型库窗口

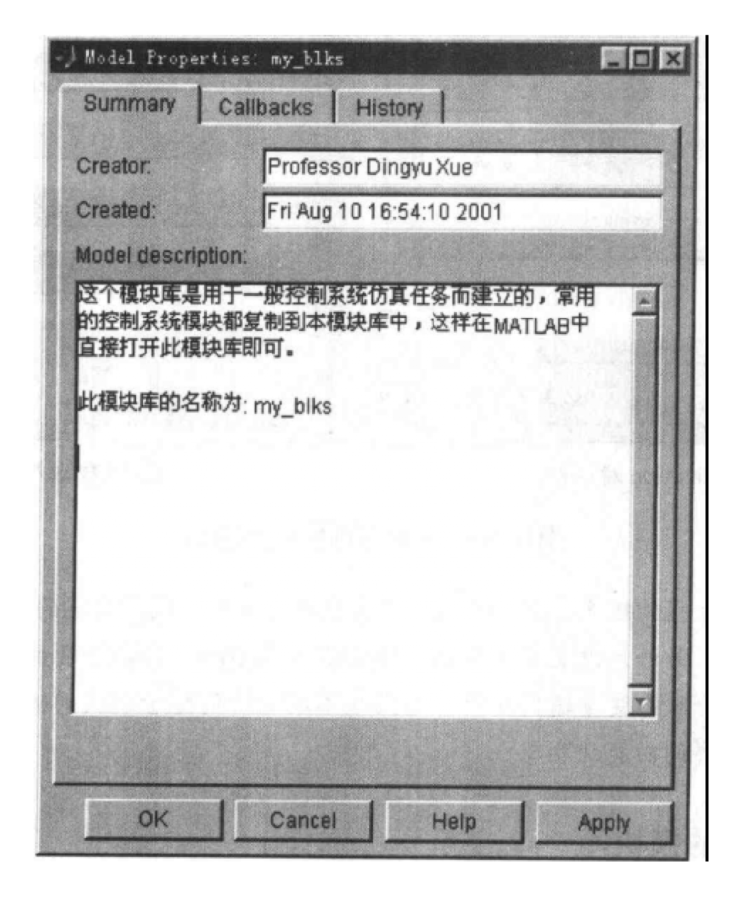


图 5-71 模型属性输入对话框

这样建立起来的模块库是处于锁定状态 (locked)，模块库中各个模块是不能进行修改或移动位置的，要想修改其中的模块，则需要使用 Edit | Unlock library 菜单项将其解锁，这样就能够修改内部模块了。修改完成后存盘，则模块库又变回锁定状态了。

有了这样的模块库，则可以将常用的一组模块复制到该模块库中，这样就能构造出如图 5-72 所示的自己的模块库，所以再绘制简单的框图时不必再打开 Simulink，而只需在 MATLAB 命令窗口下启动 my_blks 即可。

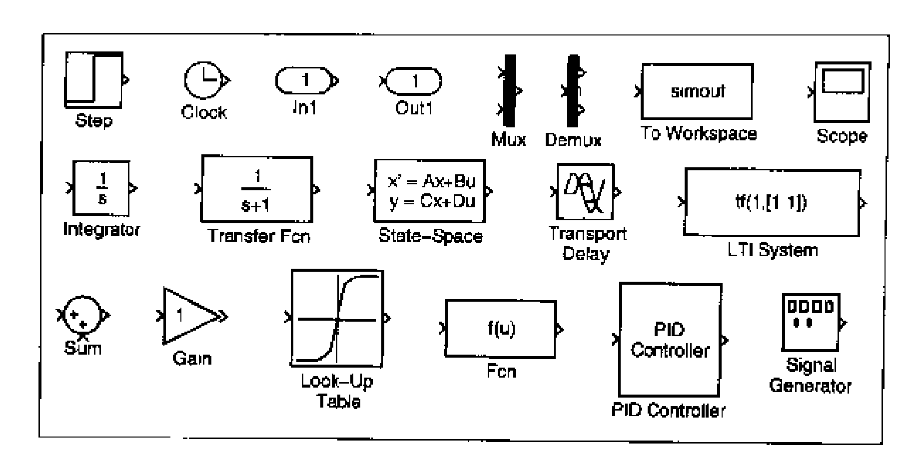


图 5-72 常用模块库窗口

5.3.5 子系统应用举例—— F14 战斗机仿真

在介绍子系统和封装技术之前先看一个复杂系统的例子，该例子是在控制系统计算机辅助设计软件不是很发达的时候由美国学者 Dean Friderik 等人提出的，当时是用于测试计算机辅助设计软件功能和建模准确性的，这就是著名的 F-14 战斗机模型与仿真的基准测试问题 (benchmark problem)[11]。该问题提出以来，国际上很多学者用不同的算法和计算机软件陆续提出了该问题的求解方法，但有些求解方法现在看来是很烦琐的，因为已经有了新一代的计算机软件，如 Simulink 这样的交互绘图式的软件来表示复杂的系统模型。

F-14 战斗机基准问题的框图如图 5-73 所示，该系统共有两路输入信号，其向量表示为 $\boldsymbol{u}=[n(t),\alpha_c(t)]^T$，其中 $n(t)$ 为单位方差的白噪声信号，而 $\alpha_c(t)=K\beta(\mathrm{e}^{-\gamma t}-\mathrm{e}^{-\beta t})/(\beta-\gamma)$ 为攻击角度命令输入信号，这里 $K=\alpha_{c_{\max}}\mathrm{e}^{\gamma t_m}$，且 $\alpha_{c_{\max}}=0.0349$, $t_m=0.025$, $\beta=426.4352$, $\gamma=0.01$。已知系统中各个模块的参数为：

$$\tau_a=0.05,\ \sigma_{wG}=3.0,\ a=2.5348,\ b=64.13$$
$$V_{\tau_0}=690.4,\ \sigma_\alpha=5.236\times10^{-3},\ Z_b=-63.9979,\ M_b=-6.8847$$
$$U_0=689.4,\ Z_w=-0.6385,\ M_q=-0.6571,\ M_w=-5.92\times10^{-3}$$
$$\omega_1=2.971,\ \omega_2=4.144,\ \tau_s=0.10,\ \tau_\alpha=0.3959$$
$$K_Q=0.8156,\ K_\alpha=0.6770,\ K_f=-3.864,\ K_F=-1.745$$

可以用下面语句定义一系列 MATLAB 变量，其顺序与前面变量列表完全对应。

```
tA=0.05; Swg=3.0; a=2.5348; b=64.1300;
Vto=690.4; Sa=0.005236; Zb=-63.9979; Mb=-6.8847;
U0=689.4; Zw=-0.6385; Mq=-0.6571; Mw=-0.00592;
w1=2.971; w2=4.144; ts=0.1; ta=0.3959;
KQ=0.8156; Ka=0.677; Kf=-3.864; KF=-1.7450;
g=0.01; be=426.4352; tm=0.025; K=0.0349*exp(g*tm);
```

最后一行语句中，变量 g 和 be 分别对应前面的 γ 和 β。可以将上述的变量赋值语句存成一个 MATLAB 文件 c5f14dat.m，这样在进行仿真之前就应该运行此文件，将这些变量在

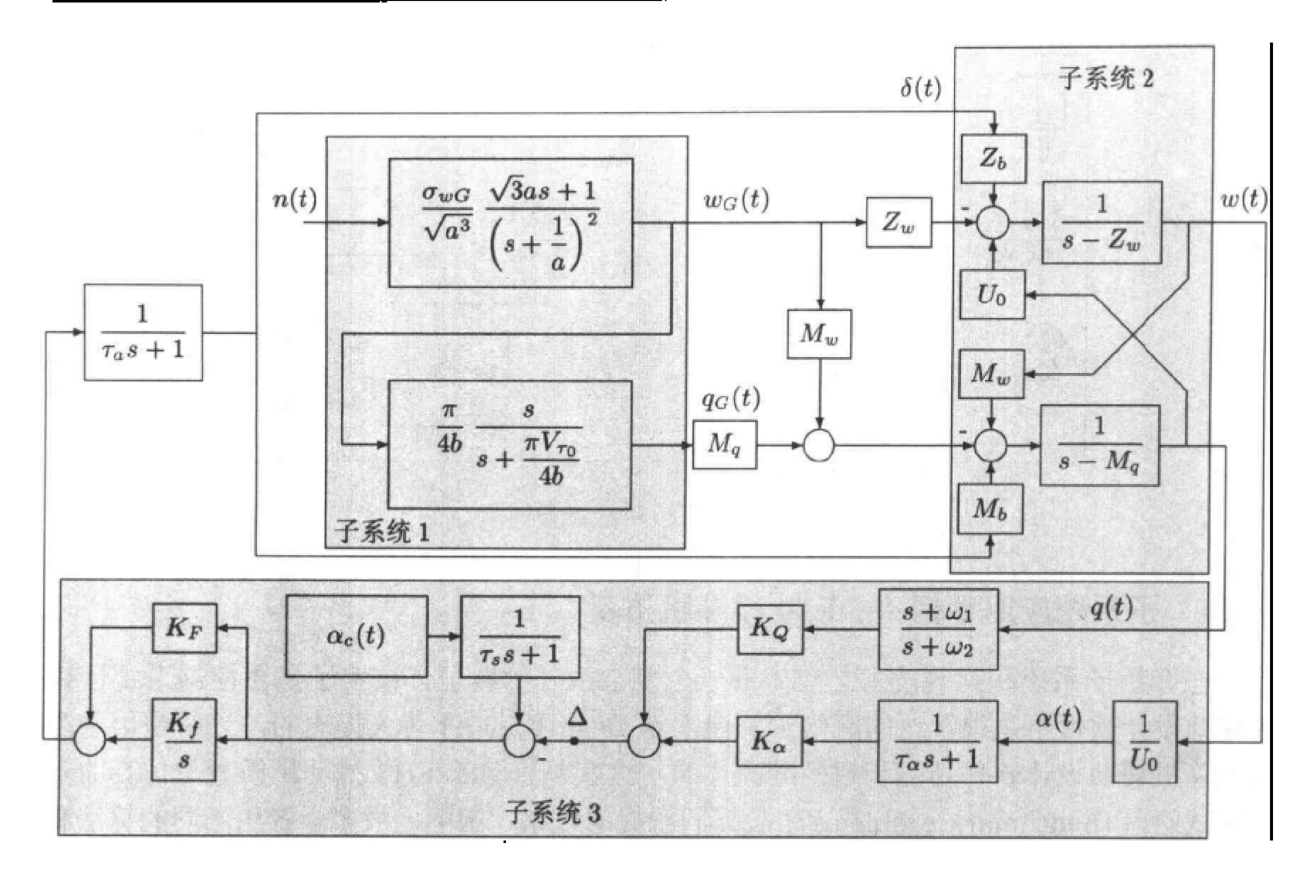


图 5-73　F14 战斗机模型框图

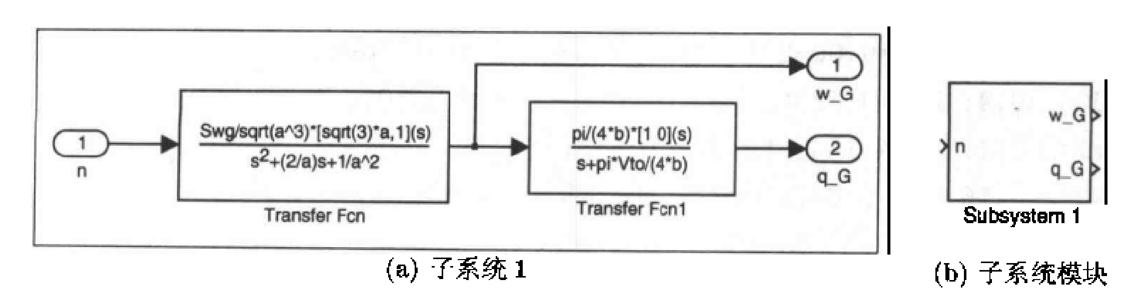
MATLAB 的工作空间内全部赋值。

观察原系统可见，整个系统的输出有三路信号，$y(t) = [N_{Z_p}(t), \alpha(t), q(t)]^T$，这里 $N_{Z_p}(t)$ 信号定义为

$$N_{Z_p}(t) = \frac{1}{32.2}[-\dot{w}(t) + U_0 q(t) + 22.8\dot{q}(t)] \tag{5.4}$$

可见原系统结构较复杂，可以将其分成 4 个子系统，其中前三个如图 (5-73) 所示，第 4 个子系统描述式 (5.4) 中给出的信号变化。

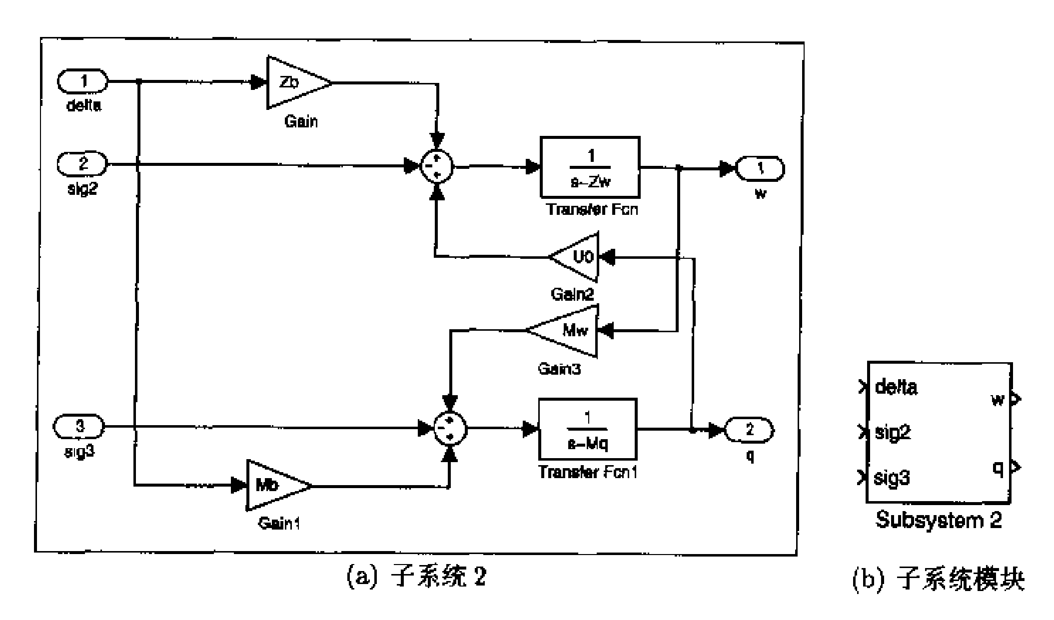
首先考虑"子系统 1"，可见该模块有 1 路输入 —— $n(t)$，有两路输出信号 —— $w_G(t)$ 和 $q_G(t)$，所以可以如图 5-74 (a) 建立其子系统模型。选中该窗口中的所有模块，再选择菜单项 Edit | Create Subsystem，则可以得出如图 5-74 (b) 所示的子系统图标。



(a) 子系统 1　　(b) 子系统模块

图 5-74　子系统 1 的 Simulink 表示

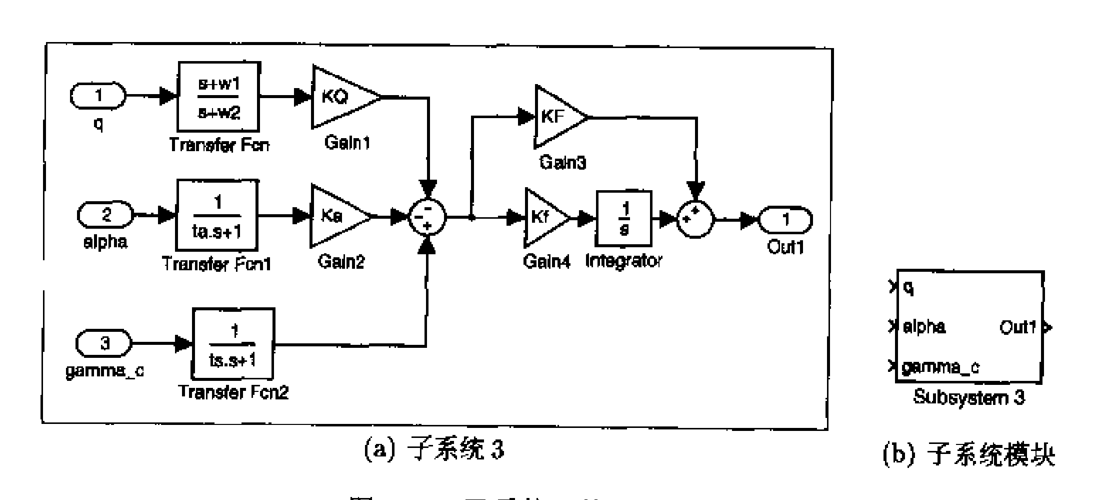
再考虑“子系统 2”，该系统由三路输入信号，其中第 1 路为 $\delta(t)$，第 2 路信号 $w_G(t)$ 经 Z_w 模块后的信号，第 3 路为 $-(M_w w_G(t) + M_q q_G(t))$，子模块有两路输出，$w(t)$ 和 $q(t)$。这样就可以建立起“子系统 2”的模型，如图 5-75 (a) 所示。制成子系统后图标如图 5-75 (b) 所示。



(a) 子系统 2　　(b) 子系统模块

图 5-75 子系统 2 的 Simulink 表示

子系统 3 有两路输入信号，$w(t)$ 和 $q(t)$，一路输出信号，故可以建立起如图 5-76 (a) 所示的子系统模型，制成子系统后图标如图 5-76 (b) 所示。

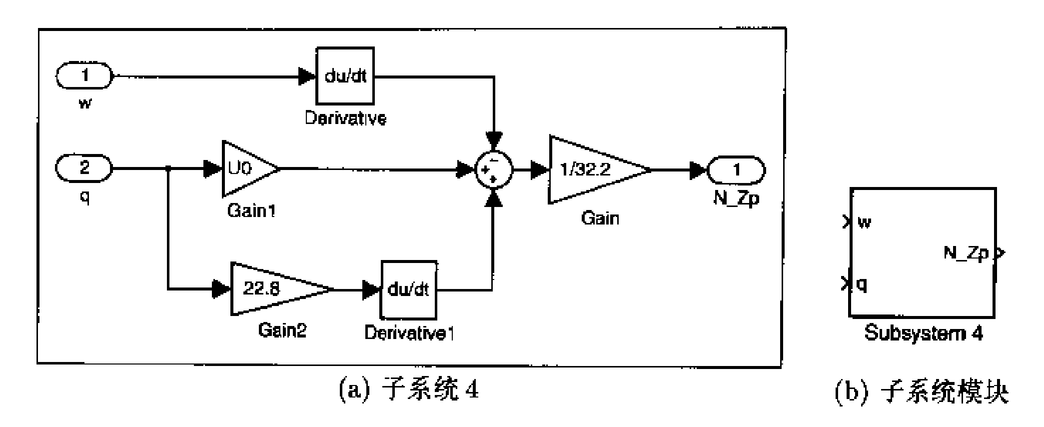


(a) 子系统 3　　(b) 子系统模块

图 5-76 子系统 3 的 Simulink 表示

现在来建立子系统 4，即用子系统的方式来表示式 (5.4)，可以由图 5-77 (a) 中所示的方式来表示该子系统，并将其表示成子系统图标形式，如图 5-77 (b) 所示。

建立起各个子系统模型后，可以比较容易地建立起整个 F-14 战斗机系统的 Simulink 模型，如图 5-78 所示。可以看出，采用子系统的形式可以使得原来很复杂的问题分解成各个相对简单的小块，然后将这些小块再连接成整个系统，这使得整个系统的建立和维



(a) 子系统 4　　(b) 子系统模块

图 5-77　子系统 4 的 Simulink 表示

护都变得更加简单。

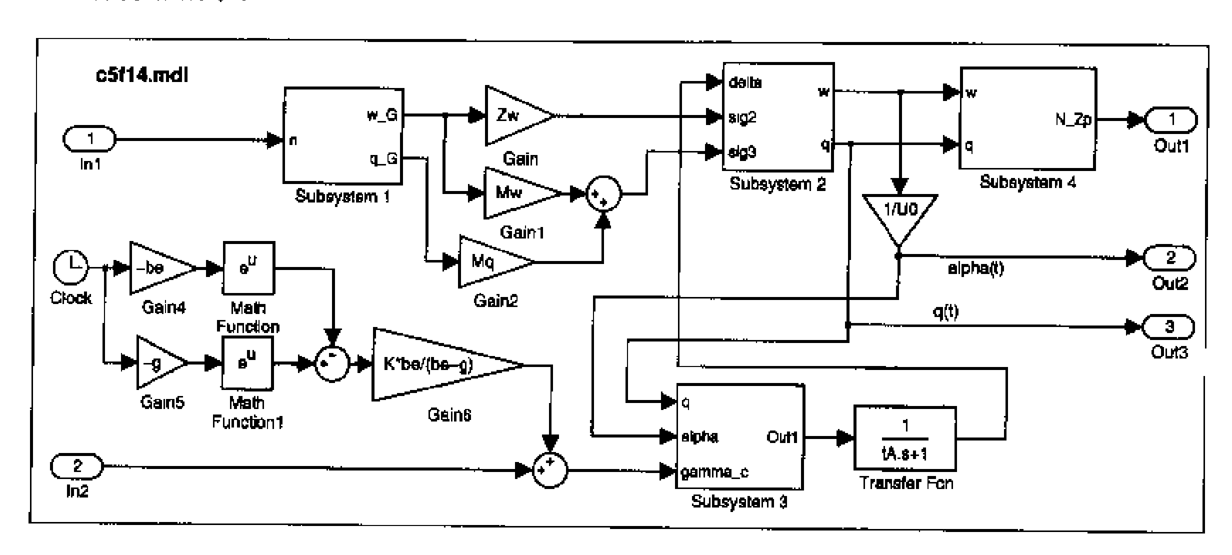


图 5-78　F-14 系统的 Simulink 模型

5.4　电力系统模块集与电子线路仿真

5.4.1　电力系统模块集简介

Simulink 中可以使用的电力系统仿真模块集 (Power Systems Blockset) 主要是由加拿大的 HydroQuébec 和 TECSIM International 公司共同开发的，其功能非常强大，可以用于电路、电力电子系统、电机系统、电力传输等过程的仿真，它提供了一种类似电路建模的方式进行模型绘制，在仿真前将自动将其变化成状态方程描述的系统形式，然后才能在 Simulink 下进行仿真分析。

在 MATLAB 命令窗口中键入 `powerlib`，则将得出如图 5-79 所示的模块集。当然，电力系统模块集中的器件还可以从 Simulink 模块浏览窗口中直接启动。可见，在该模块集中还有很多子模块集，双击每一个图标都将打开一个下级子模块集，例如双击

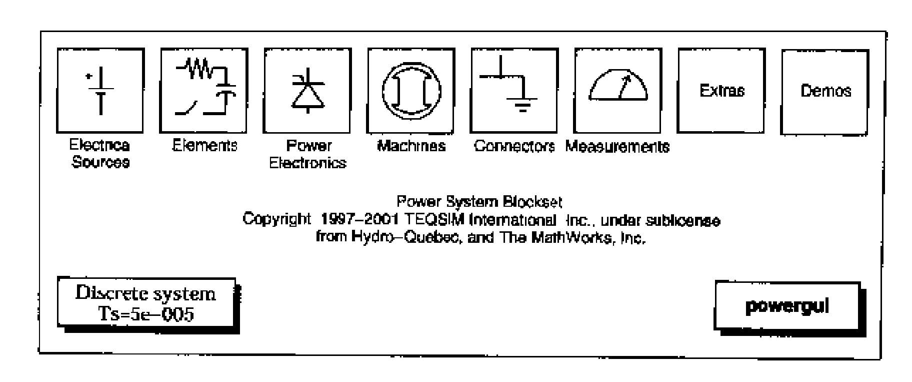
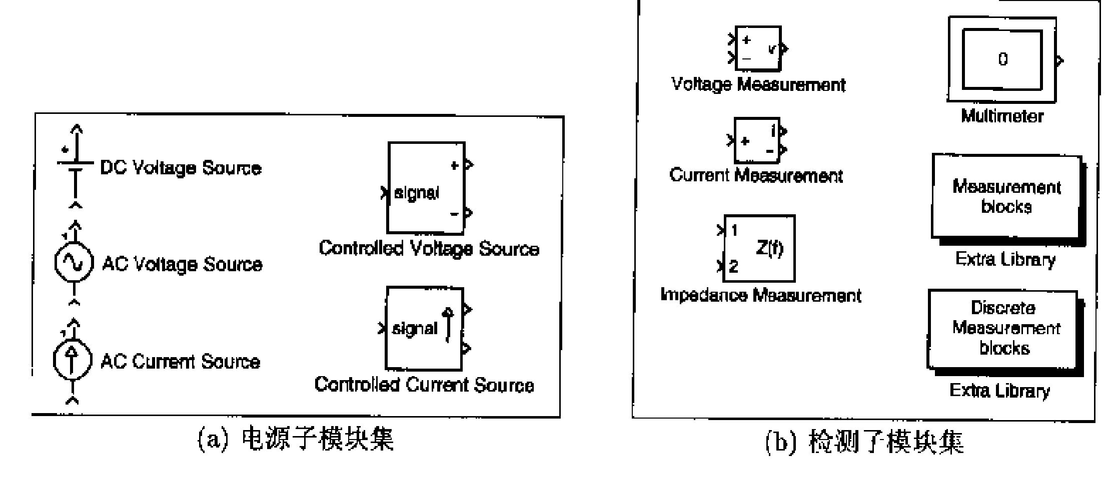


图 5-79 电力系统模块集

Electrical sources 图标将打开如图 5-80 (a) 所示的电源子模块集，其中有直流和交流电



(a) 电源子模块集　　(b) 检测子模块集

图 5-80 电力系统模块集的输入与输出

源，以及各种受控电流源和电压源等。若双击模块集中的 Measurements 图标，则将得出如图 5-80 (b) 所示的子模块集，其中有各种检测端口，如电流表、电压表和阻抗表，该组中还包括各种其他扩展的子模块集。

这里将分别介绍不同的电力系统建模仿真问题的求解方法，如电路的仿真、功率电子系统的仿真和电机系统的仿真等。

5.4.2 电路的仿真

我们知道，电路中最常用的元件是电阻、电容和电感，双击电力系统模块集中的 Elements 图标，则将得出如图 5-81 所示的子模块集，其中既包含各种电阻、电容和电感元件，还包含各种变压器元件，另外还有一个三相元件子模块集。

从普通的电阻、电容和电感元件来看，有串联的 RLC (电阻、电感、电容) 分支和并联的 RLC 分支，以及它们的负载形式。

双击 Series RLC Branch (串联 RLC 分支) 元件，则将得出如图 5-82 所示的对话框，在这个对话框中适当地输入电阻、电容和电感的参数即可。注意，和以前介绍的纯数字

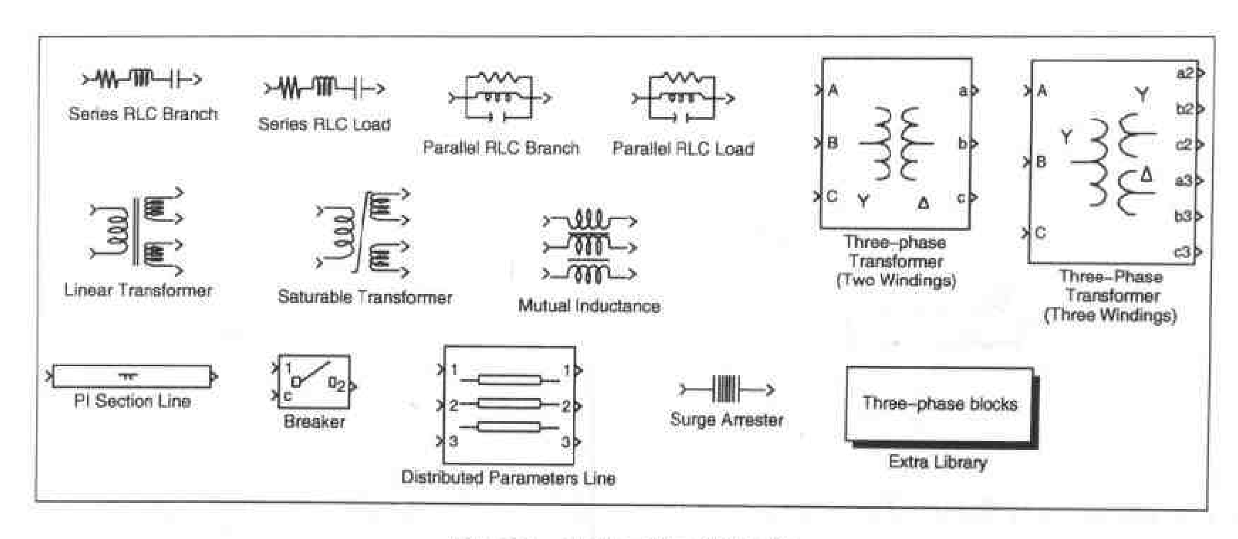


图 5-81 电路元件子模块集

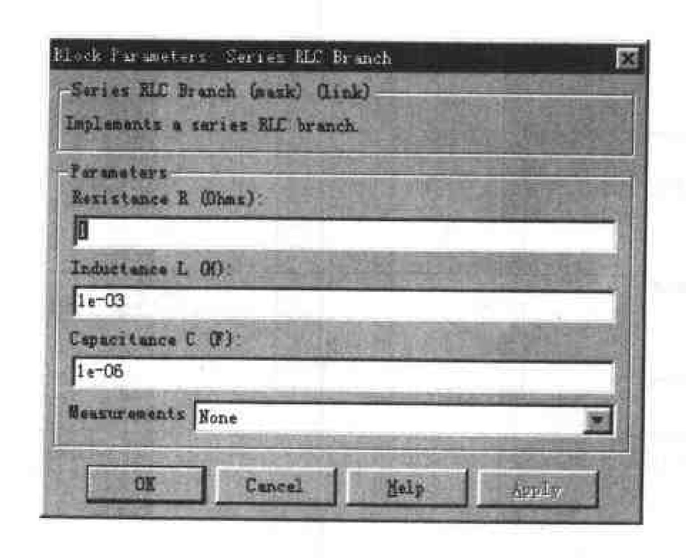


图 5-82 串联 RLC 分支元件参数对话框

仿真不同，这里填写电路参数时应该注意其单位。

遗憾的是这里不包含单个的电阻、电感和电容元件，可以从串联或并联的分支来定义单独的电路，但在串联或并联分支中直接删除某个元件也不是太容易的事，例如在串联分支中删除电容，则不能将其数值填写为 0，而需要写成 **inf**。单个电阻、电感、电容元件的参数设置在串联和并联分支中是不同的，具体请参见表 5-1。为了搭建实验的方便，也可以按该表拆分出单个元件，封装起来。

在一般电路中，除了前面介绍的一些元件外，还需要一些连线器类模块，双击电力系统模块集中的 Connectors 图标则将打开如图 5-83 所示的连线器类子模块集。

〖例 5.20〗 考虑如图 5-84 (a) 所示的感应电机的等效电路，输入的交流电压源为 220V, 50Hz, 其他参数值为 $R_1 = 0.428\Omega$, $L_1 = L_2 = 1.926\text{mH}$, $R_2 = 1.551\Omega$, $R_3 = 1.803\Omega$, $L_3 = 31.2\text{mH}$, $R_3 = 1.803\Omega$。按图 5-84 (b) 的方式将所需的电路元件复制到模型编辑窗口中。注意，在该框图的底部将从连接器子模块集中复制两路合并的元件。有了这些元件后，可以分别对各个元件进行

表 5-1 单个电阻、电感、电容参数设置表

元件	串联 RLC 分支			并联 RLC 分支		
类型	电阻数值	电感数值	电容数值	电阻数值	电感数值	电容数值
单个电阻	R	0	inf	R	inf	0
单个电感	0	L	inf	inf	L	0
单个电容	0	0	C	inf	inf	C

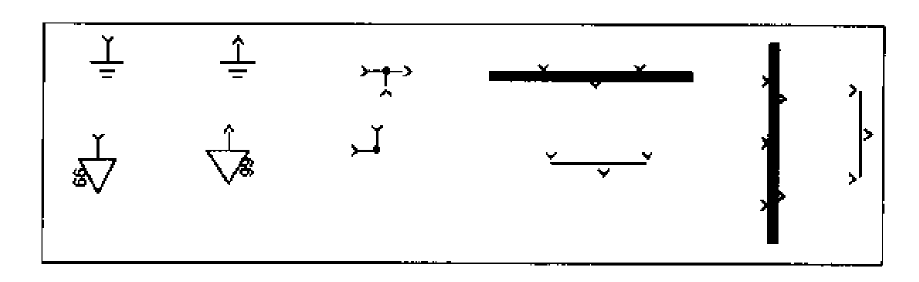
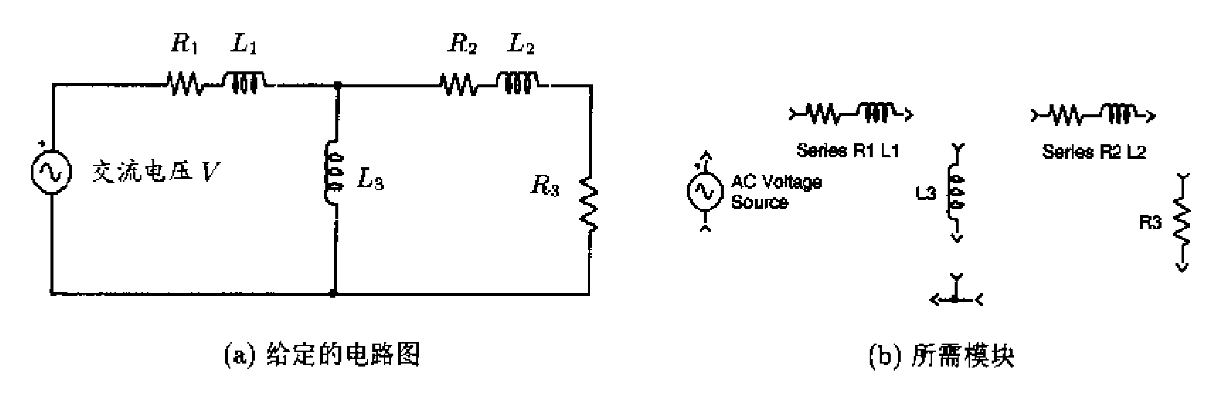


图 5-83 连线器类子模块集



(a) 给定的电路图 (b) 所需模块

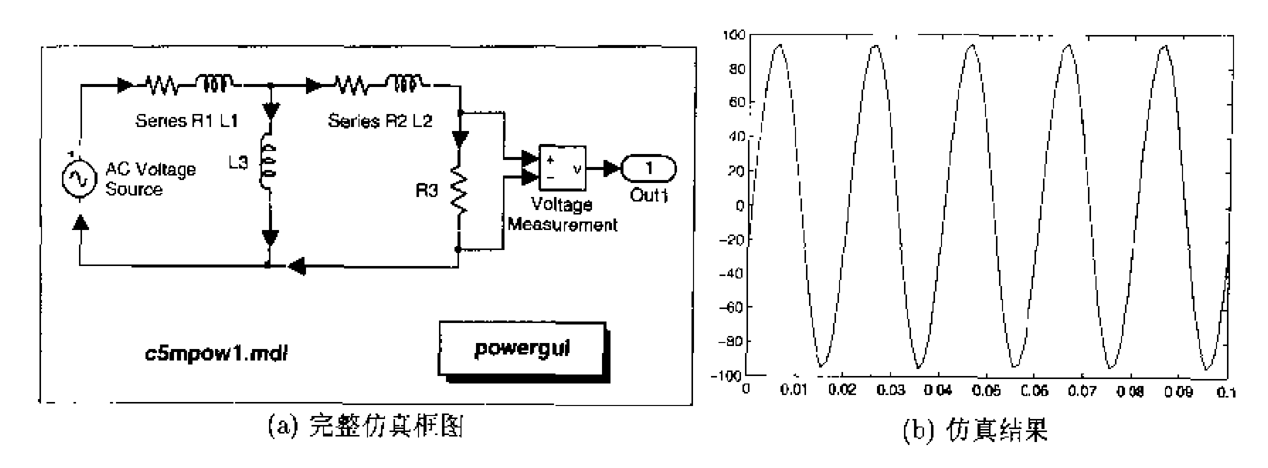
图 5-84 简单电路的 Simulink 模块搭建

赋值，例如 Series R1 L1 模块的三个参数分别为 0.428, 1.926e-03 和 inf。则可以连接整个电路了，连接完成后还应该在电路的输出端加一个电压测量元件，并将输出送给普通的 Out1 模块，最终得出如图 5-85 (a) 所示的系统框图。还应该双击电压源对话框，在其中填写适当的参数。

完成了电路连接后，将仿真的终止时间设置成 0.1，则可以启动仿真过程。开始仿真过程后，则将在 MATLAB 的命令窗口中显示如下的信息

```
Power System Blockset processing c5mpow1 ...
Computing state-space representation of linear electrical circuit (V2.2)...
(2 states ; 1 inputs ; 1 outputs)
Computing steady-state values of currents and voltages ...
Build the Simulink equivalent circuit ...
(Circuit stored inside "c5mpow1/Voltage Measurement" block)
Ready.
```

该信息表明系统自动完成并通过了从绘制的电路图到状态方程的转换，开始自动启动状态



(a) 完整仿真框图　　(b) 仿真结果

图 5-85　简单电路仿真

方程的仿真过程，仿真结束后，和以前介绍的仿真过程一样，将在 MATLAB 工作空间中生成两个变量，tout 和 yout，用 plot(tout,yout) 语句就能绘制出系统的仿真结果，如图 5-85 (b) 所示。从仿真结果曲线看，该系统的输出基本上是正弦波信号，这完全是由输入交流电压的 50Hz 振荡引起的正常现象。

还可以对仿真电路进行稳态分析。将电力系统模块集中的 powergui 模块复制到仿真框图中，则双击该模块就可以对电路图中的信号进行稳态分析，得出如图 5-86 所示的分析结果。从得出的结果中可以看出所量测的信号稳态曲线为幅值 95.27V，初始相位为 $-15.33°$ (拖动滚动杆才能显示出来) 的正弦信号，亦即，该信号的解析表达式在稳态时趋近 $95.27\sin(2\pi \times 50t - 15.33°)$。事实上，如果使用下面的 MATLAB 语句将得出两条几乎重合的曲线。

```
>> y1=95.27*sin(2*pi*50*tout-15.33*pi/180); plot(tout,yout,tout,y1)
```

另外，用电力系统工具箱中提供的 power2sys() 可以提取出从给定电源到输出端子的状态方程模型，根据此状态方程模型就可以对整个电路进行频域分析，如绘制其 Bode 图等。power2sys() 函数的调用格式为：

```
[a,b,c,d]=power2sys(模型名)
```

其中 a, b, c, d 为系统的状态方程矩阵，由 tf() 还可以得出系统的传递函数模型。

【例 5.21】 对前面给出的电路模型，可以用下面的命令对系统进行下列分析

```
>> [a,b,c,d]=power2sys('c5mpow1')  % 获得系统的状态方程
a =
  -114.4379 -844.6462
  -107.7843 -896.7868
b =
   267.3783
   251.8325
c =
```

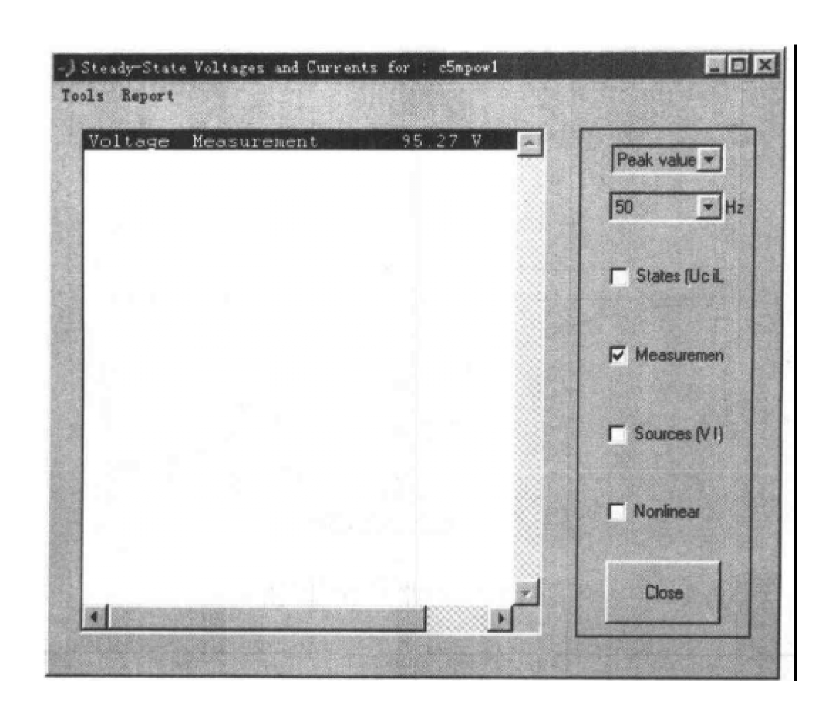


图 5-86 电路稳态分析界面

```
            0    1.8030
d =
            0
>> G=ss(a,b,c,d); bode(G) % 绘制系统的 Bode 图
   G1=tf(G)  % 获得等效传递函数模型
Transfer function:
  454.1 s - 5.164e-011
------------------------
s^2 + 1011 s + 1.159e004
```

由上面语句获得的系统 Bode 图如图 5-87 所示，可见该模型有带通特性。

5.4.3 功率电子系统仿真

电力系统模块集中提供了功率电子学系统仿真的功能，双击模块集中的 Power Electronics 图标，则将得出如图 5-88 所示的子模块集，在该模块组中提供了二极管 (Diode)，晶闸管模块 (Thyrister)，可关断可控硅 (GTO)，场效应管 (MOSFET)，绝缘栅二极管 (IGBT) 等模块。可以注意到，这里每个模块均有一个 m 输出端子，从该端子可以到处模块内部所有的信号，该信号可以直接连接到 Simulink 的输出模块上。

〖例 5.22〗 电力系统模块集不但能仿真简单的电路，而且也能仿真功率电子学领域的系统。假设有如图 5-89 所示的场效应管电路，在该电路中有一个功率电子元件 (MOSFET)，一个普通二极管，另外，此电路中有电阻、电感、电容元件，有直流电压源和一个电流源。

可以将这些元件从相应的模型库中复制到一个空白的模型窗口中，并搭建起如图 5-90 所示的

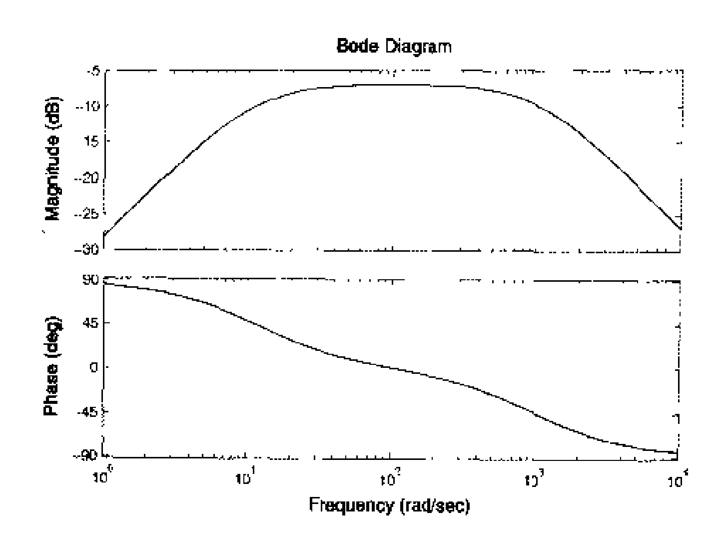


图 5-87 电路的 Bode 图

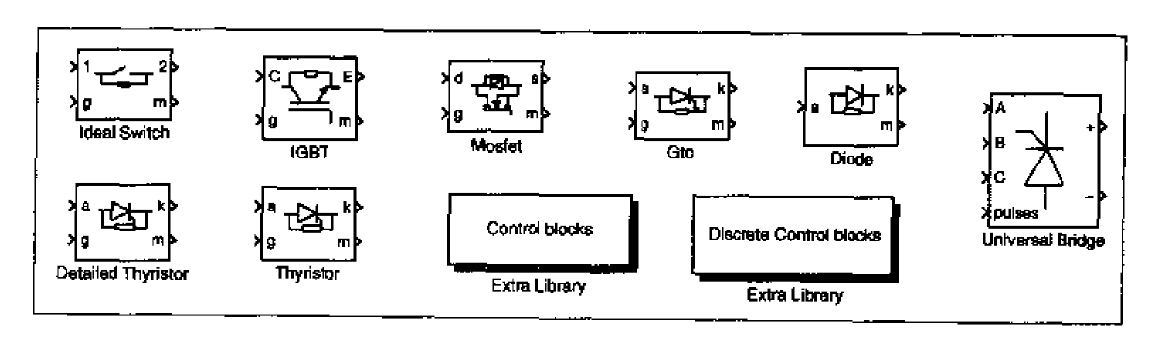


图 5-88 功率电子子模块集

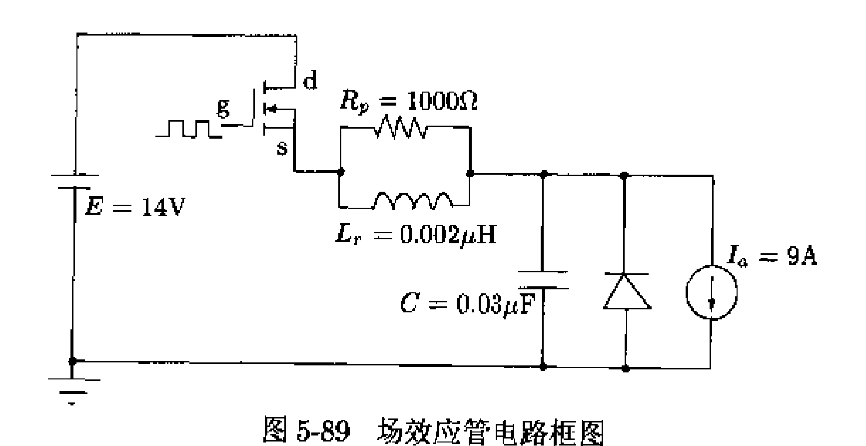


图 5-89 场效应管电路框图

Simulink 电路。在该电路中不但有上述的元件，还有一个测电压的元件测量电容两端的电压。

另外，电力系统中提供的电子元件 MOSFET 和 Diode 都带有一个 m 输出端口，该端口将返回相应的元件内部的 Simulink 信号，例如，二极管输出的 m 信号为 $[I_{ak}, V_{ak}]$，即二极管的电流和端电压。这里只对其内部电流感兴趣，所以用了一个选路器选择其第 1 路信号，将其接为输出端子的第 4 路输入。场效应管的 m 信号为 $[I_d, V_{ds}]$，其中 I_d 为流入 d 端子的电流，V_{ds} 为 d-s 端间的电压，仍用选路器将 I_d 信号接入输出端子的第 2 输入端。

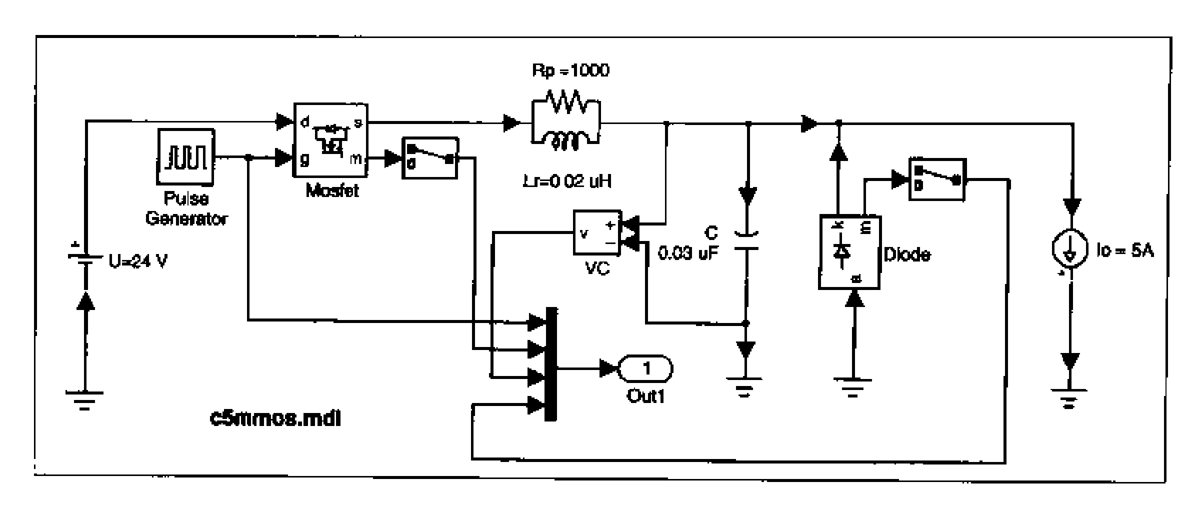
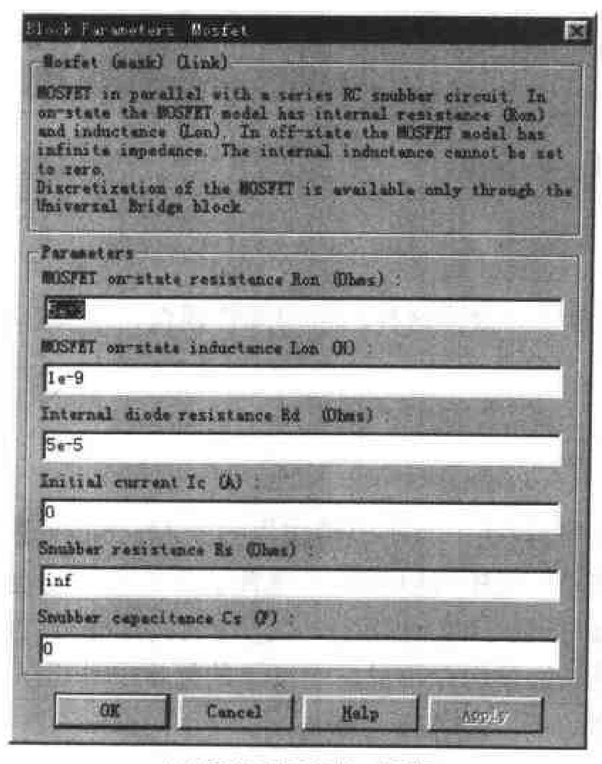
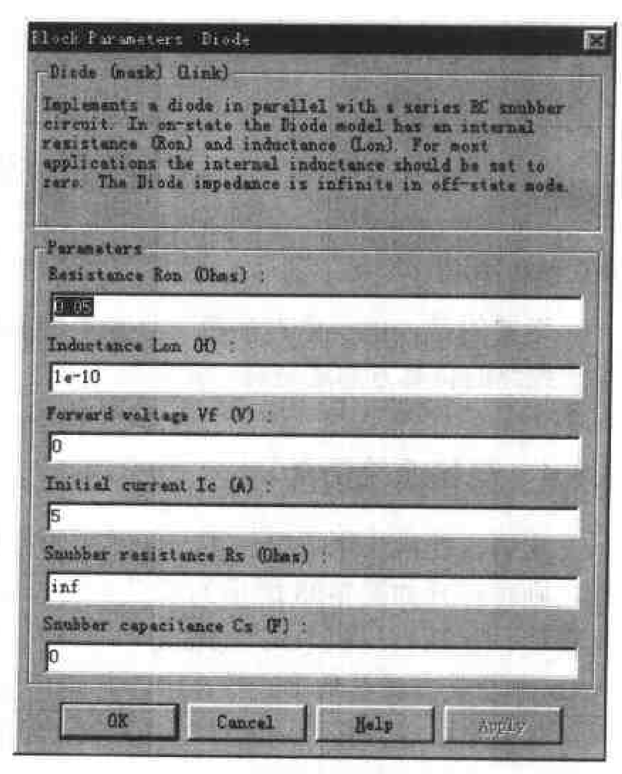


图 5-90 场效应管电路的 Simulink 框图

场效应管的正常工作应该依赖于其门极 g 的触发信号，在该端可以加一个周期为 0.5μs，脉宽 20%，幅值为 1 的脉冲信号发生器。另外应该按图 5-91 (a) 和 5-91 (b) 所示的对话框输入场效应管和二极管的参数。



(a) 场效应管参数对话框



(b) 二极管参数对话框

图 5-91 相关模块设置对话框

选择仿真终止时间为 2μs，亦即 2e-6 秒，并选择 ode15s 仿真算法，则可以启动仿真过程，得出的仿真结果如图 5-92 所示。

从得到的仿真结果可见，可以通过这样简单的方法直接"观察"一小段时间内电路中的信号，这是一般硬件实验所难以实现的，所以对电力电子系统进行"软实验"有其普通硬件实验无

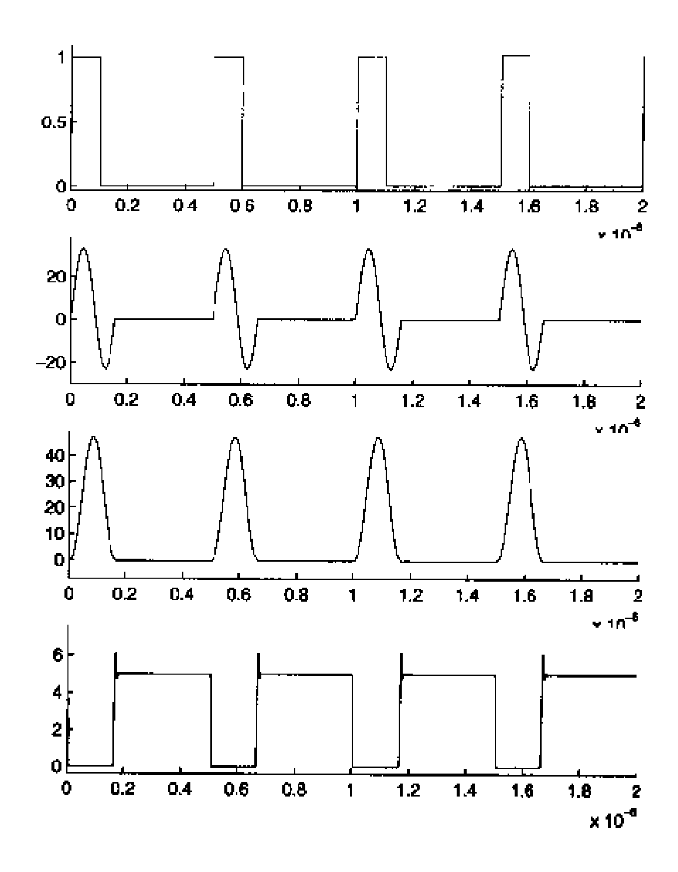


图 5-92 电路仿真结果

法企及的优势。

遗憾的是，由于电力系统工具箱不能有效地给三极管这样的常用元件建模，所以在电子线路的仿真方面还是有一定的局限性的。

5.4.4 电机系统仿真

电力系统模块集还提供了一些常用的电机仿真模块，双击该模块集中的 Machines 图标，则将打开如图 5-93 所示的子模块集，从该模块组中可以看出，包含了各种各样的电机模型，包括直流机 (DC motors)、异步机 (Asynchronous motors)、同步机 (Synchronous motors) 及其各种其他形式等，限于篇幅，这里只能举例介绍星型连接下的异步电动机仿真，其他电机的仿真可以具体参考有关文献 [18]。

〖例 5.23〗 电力系统工具箱中提供了两个异步电动机模型，其图标表示如图 5-94 (a) 所示。左侧的为标幺值单位下的异步电动机 (p.u.，即 per unit) 模型，右侧的为国际单位制 (SI) 下的异步机模型。在本例中主要考虑国际单位制的异步电动机模型，所以可以将该模型复制到空白的模型编辑窗口中。

异步电动机模块有 4 个输入端子，4 个输出端子，前 3 个输入端子 (A, B, C) 为电机的定子电压输入，一般可直接接三相电压，可以有星型和三角形两种解法，第 4 输入端一般接负载，为轴上的机械转矩，该端子可以直接接 Simulink 信号。模块的前 3 个输出端子 (a, b, c) 为转子电压输

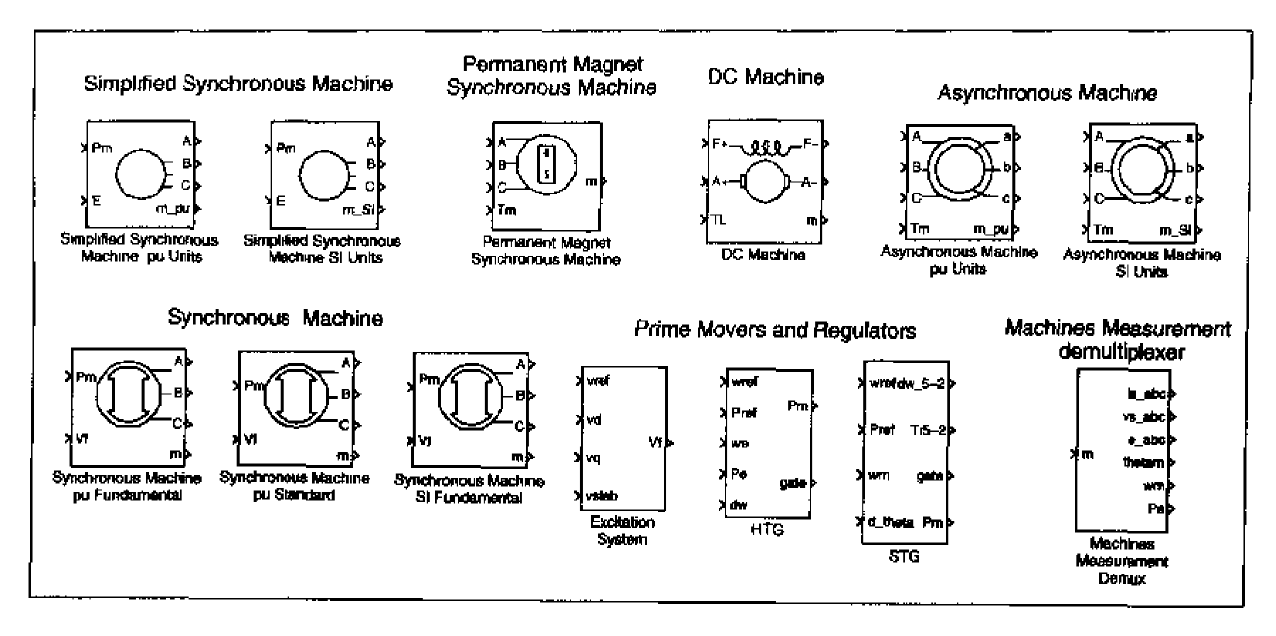
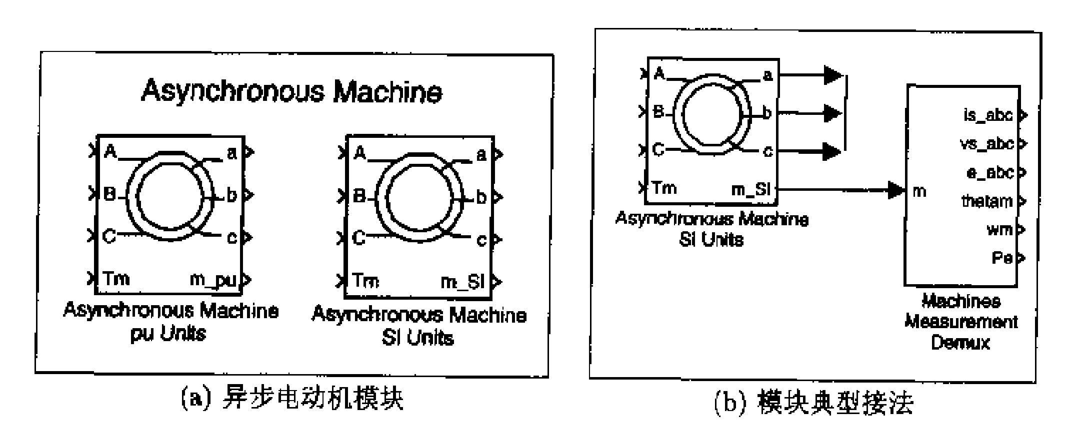


图 5-93 电机子模块集



(a) 异步电动机模块 (b) 模块典型接法

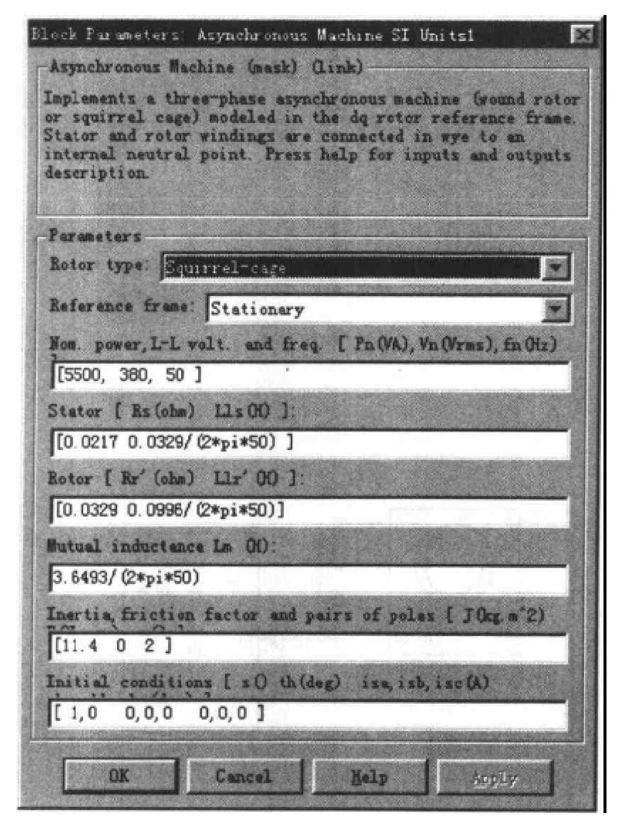
图 5-94 异步电动机模块

出，一般短接在一起，或连接到其他附加的电路中。第 4 个输出端为 m 端子，它返回一系列电机的内部信号集，共有 21 路信号组成，其构成如下：

- 第 1 到第 3 路：转子电流 $i'_{\rm ra}$，$i'_{\rm rb}$，$i'_{\rm rc}$；
- 第 4 到第 9 路：q-d-n 坐标系下的转子信号，依次为 q-轴电流 $i'_{\rm qr}$，d-轴电流 $i'_{\rm dr}$，q-轴磁通 $\psi'_{\rm qr}$，d-轴磁通 $\psi'_{\rm dr}$，q-轴电压 $v'_{\rm qr}$，d-轴电压 $v'_{\rm dr}$；
- 第 10 到第 12 路：定子电流 $i_{\rm sa}$，$i_{\rm sb}$，$i_{\rm sc}$；
- 第 13 到第 18 路：q-d-n 坐标系下的定子信号，依次为 q-轴电流 $i_{\rm qs}$，d-轴电流 $i_{\rm ds}$，q-轴磁通 $\psi_{\rm qs}$，d-轴磁通 $\psi_{\rm ds}$，q-轴电压 $v_{\rm qs}$，d-轴电压 $v_{\rm ds}$；
- 第 19 路到第 21 路：电机转速 $\omega_{\rm m}$，机械转矩 $T_{\rm m}$，电机转子角位移 $\theta_{\rm m}$。

该路信号应该接电机测试信号分路器 (Machines Measurement Demux) 模块 将各路信号分离出来，以便直接接示波器或 Simulink 输出端子进行显示。通常可以按照图 5-94 (b) 中所示的方式连接模块的输出端。

双击异步电动机模块，将得出该模块的参数对话框，如图 5-95 (a) 所示。在该对话框中需要



(a) 异步电动机参数对话框

(b) 分路器设置对话框

图 5-95 相关模块设置对话框

输入如下参数：

- 绕组类型 (Rotor type) 列表框：分为绕线式 (Wound) 和鼠笼式 (Squirrel-cage) 两种，后者将不显示输出端 a, b, c，而直接将其在模块内部短接。
- 参考坐标系 (Reference frame) 列表框：有静止坐标系 (Stationary)，转子的 (Rotor) 和同步的 (Synchronous)，一般常选择静止坐标系；
- 额定参数：额定功率 P_n (单位：kw)，线电压 V_n (单位：V)、频率 f_n (单位：Hz)；
- 定子电阻 R_s' (Stator) (单位：Ohms) 和漏感 (L_{ls}') (单位：H)；
- 转子电阻 R_r (Rotor) (单位：Ohms) 和漏感 (L_{lr}) (单位：H)；
- 互感 (Mutual inductance) L_m (单位：H)；
- 转矩 (inertia) J (单位 kg·m^2)，摩擦系数 F (单位：N·m·s)，极对数。

这些参数基本上都是电动机的铭牌参数，例如已知某电动机参数为[16]:

$P_n = 5.5\text{kW}$, $V_n = 380\text{V}$, $f_n = 50\text{Hz}$,

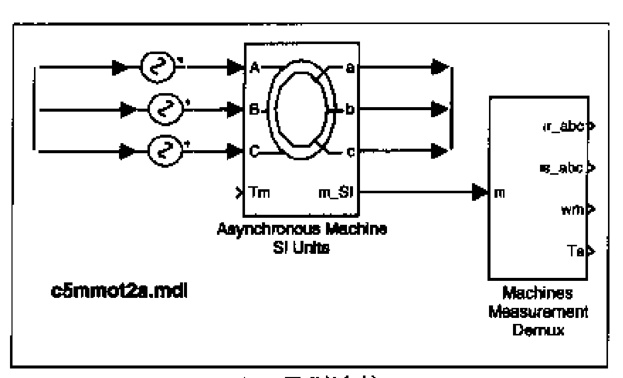
$R_s' = 0.0217\Omega$, $x_{ls}' = 0.039\Omega$, $R_r = 0.0329\Omega$, $x_{lr} = 0.0996\Omega$,

$L_m = 3.6494\Omega$, $J = 11.4\text{kg}\cdot\text{m}^2$, $F = 0$, $P = 2$。

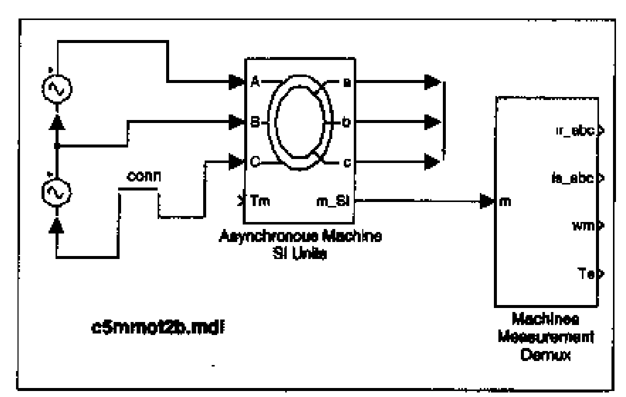
这里电感由电抗的形式给出，如果想获得电感的数值则需用 $L = x/(2\pi f)$ 公式算出。

该系统中的测量信号分路器直接使用也是有问题的，因为其默认的分路是针对简化的同步机 (Simplified synchronous) 输出的，双击该模块将得出如图 5-95 (b) 所示的对话框，在电机类型 (Machine type) 栏目中选择异步机 (Asynchronous)，则得出信号列表，可以从该信号列表中选择想输出的信号。

常用的异步电动机接法一般有星型接法和三角形接法，分别如图 5-96 (a) 和 (b) 所示。在本例



(a) 星型连接



(b) 三角形连接

图 5-96 常用电机接法

中可以采用星型接法，在该结构下，3 路输入信号的初始相位应该分别设置为 0, 120 和 240。在绘制三角形接法时，下面的交流电压源和异步机的 C 端口都是“输入”端子，可以用连线器中的器件将其连接起来，如图中的 conn，事实上，在纯电力系统工具箱模块构成的系统模型中，信号是没有方向的，所以这样构造的模型能正确进行仿真运算。

这样可以绘制出异步机仿真框图，如图 5-97 所示。这里电源采用了星型接法，可以将这三个交流电压源的电压值设成相电压 220V, A, B, C 三相的相位分别填写为 0, 120, 240。

在该模型中有一个空闲的电流测试元件 (Current Measurement)，这是电力系统工具箱所要求的，因为它要求至少有一个测量元件。另外还需要从输出的信号中使用 Selector 元件提取所需的单路信号，这样示波器中同时输出 4 路信号，a 相转子电流 i_{ar}', a 相定子电流 i_{as}，转速 ω 和输出转矩 T，设置仿真的终止时间为 3 秒，并选择仿真算法为 ode15s，相对允许误差和绝对允许误差均为 10^{-7}，就可以对系统进行仿真，得出如图 5-98 (a) 所示的仿真结果。

将 A,B 两相换序，亦即将 A 和 B 相的相位移分别改成 120 和 0，则再进行仿真就能得出如图 5-98 (b) 所示的仿真结果，可以看出，A,B 两相换序将使得电机反转。可以通过这样的方法对各种电机接法进行仿真分析。

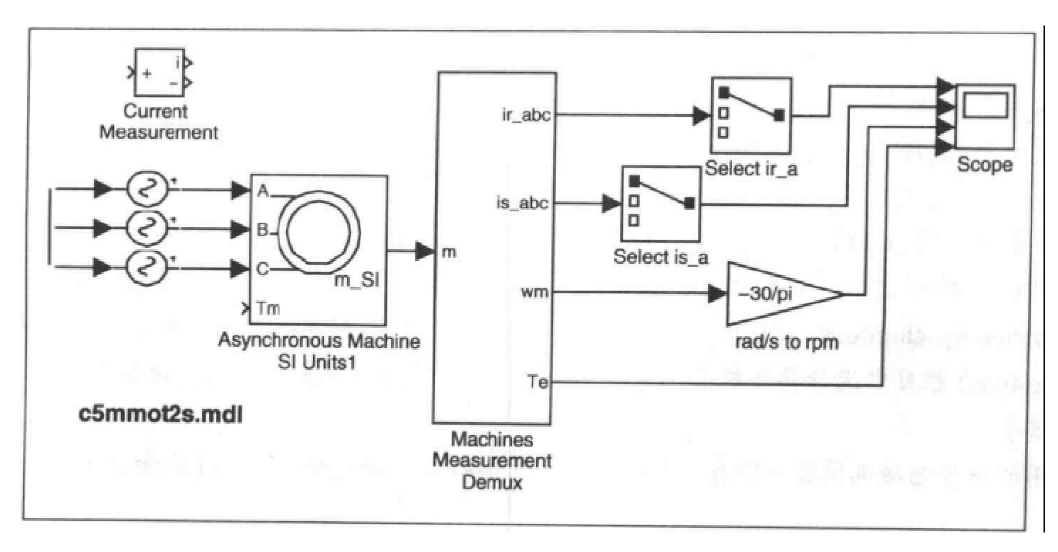
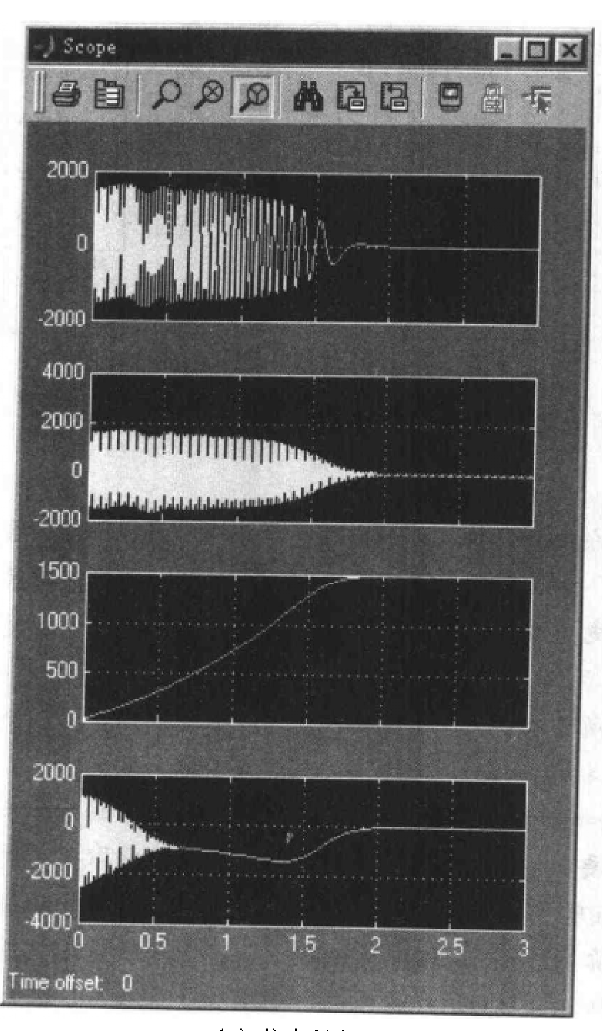
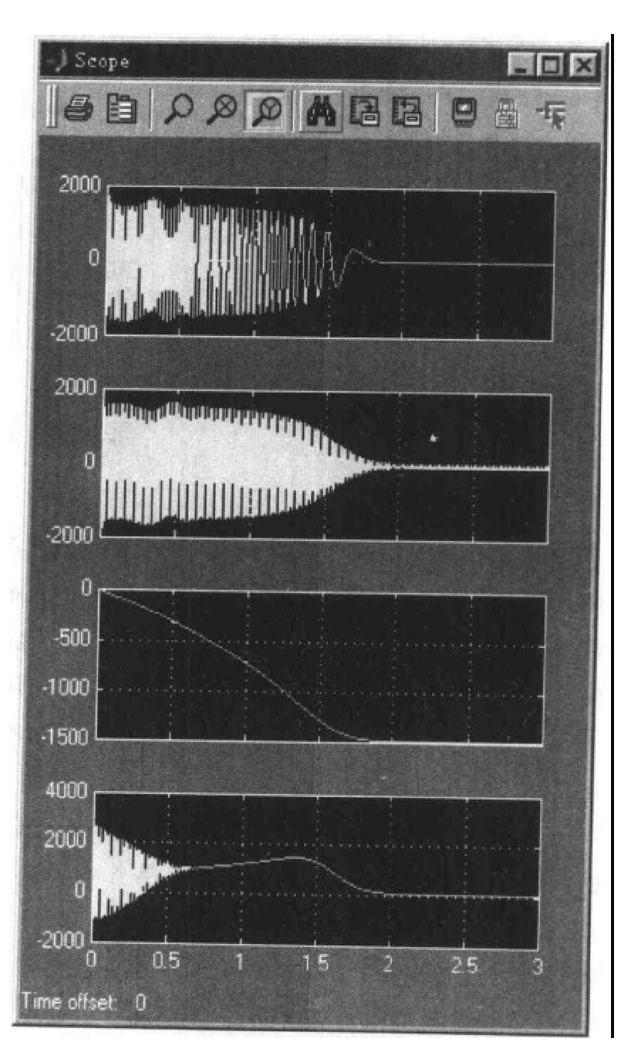


图 5-97 异步机仿真 Simulink 模型



(a) 仿真结果



(b) A,B 相换序后的仿真结果

图 5-98 异步电动机的仿真结果

5.4.5 Spice 与 Simulink 的接口

从前面的介绍可以发现，电力系统工具箱在供电、电机等系统仿真领域是很有优势的，但在一般纯电路与电子线路仿真的领域并非首选软件。Spice 语言是美国加州大学伯克利分校开发的电子线路仿真语言，PSpice 是美国 MicroSim 公司为其编写的界面，支持在 Windows 界面下的电子线路设计，是目前电子线路设计最有影响，也是最常用的软件之一[12]，在国内很多院校电子设计课程都采用该软件作为计算机辅助分析与设计工具。如果电路元件只是整个控制系统中的一个部分，那么用 Spice 就远远不够了。所以，将 Spice 电路转换为 Simulink 模块是相当必要的。本节中将介绍这样的转换程序，使得在 Simulink 下可以直接调用由 Spice 设计的电路图，进行仿真。

目前能进行 Spice 到 MATLAB/Simulink 转换的总共有两个工具，其中一个是将 Spice 文件 (.cir) 通过 MATLAB 下的动态连接技术直接运行，将结果传回到 MATLAB 工作空间。该软件的下载地址为

    http://ave.dee.isep.ipp.pt/~jcarlos/matlab

该软件是可以免费下载的，构造 DLL 文件的 C-Mex 源程序也可以免费下载，所以直接采用该模块可以对 Spice 描述的电路进行仿真，但目前看来很难将电路模块嵌入到整个 Simulink 模型中，除非对该 C-Mex 程序作适当的修改，使之允许作这样的嵌入。该软件主要由 spicemat.zip 文件构成，将该文件解到 c:\spice 目录，然后将其中的 spice.dll 文件移动到 MATLAB 的 work 目录即可。

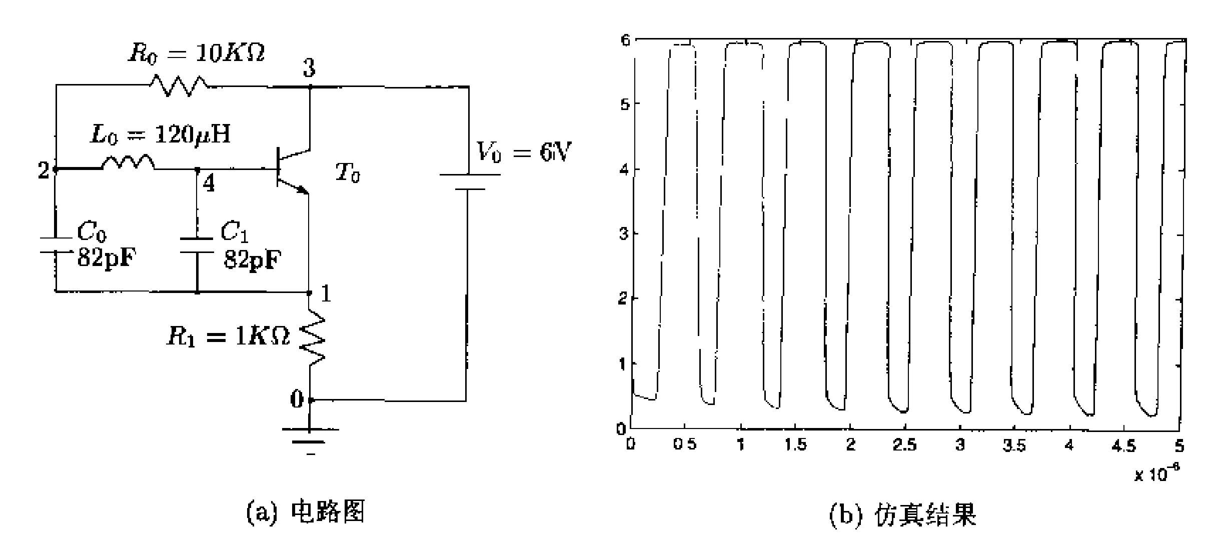
另一个软件称为 SLSP (Simulink Spice Interface)，可以将 .cir 文件嵌入到 Simulink 模型中进行仿真。显然这样的方法更具实际意义，但该软件不是免费软件。该软件发布者的地址为

    http://www.bausch-gall.de/prodss.htm

这里只能对前一种方式进行适当的介绍，并在第 6 章介绍了 S-函数后给出了将 Mex 转换成 C-函数的习题，以便读者可以自己去试试能否将其转换成可以嵌入 Simulink 的模块。

【例 5.24】 假设有一个包含三极管的电路图，如图 5-99 (a) 所示。图中所标的数字是 Spice 描述模型时所用的节点编号。如果将其存为 c5ftri1.cir 文件，并按照 MATLAB 可以调用的格式对之进行修改，则可以得出下面加注释的清单：

```
c5ftri1.cir  * 文件名
.TRAN 5ns 2us 0s 5ns UIC * 仿真参数设置，下面将叙述该句语法
   V0 3 0 DC 6V  * 电压模块，从节点 3 到 0，直流，6V
   R0 2 3 10Kohm * 电阻 R0，从节点 2 到 3，10K
   R1 1 0 1Kohm  * 电阻 R1，从节点 1 到 0，1K
   C0 4 1 82pF   * 电容 C0，从节点 4 到 1，82pF
   C1 1 2 82pF   * 电容 C1，从节点 1 到 2，82pF
   Q_npn_0 3 4 1 Qn2n_2N2222A * 三极管，接在节点 3,4,1 上，见下面 MODEL
   L0 2 4 120uH  * 电感 L0，从节点 2 到 4，120uH
```



(a) 电路图 (b) 仿真结果

图 5-99 含有三极管的电路图

```
.PRINT TRAN V(1) * 输出节点 1 的电压，以下为三极管子模型：
.MODEL Qn2n_2N2222A NPN(Is=11.6fA BF=200 BR=4 Rb=1.69ohm Re=423mohm
+Rc=169mohm Cjs=0F Cje=19.5pF Cjc=9.63pF Vje=750mV Vjc=750mV Tf=454ps
+Tr=102ns mje=333m mjc=333m VA=113V ISE=170fA IKF=410mA Ne=2)
.END
```

其中，根据 Spice 规则，瞬态分析函数 .TRAN 应该带有 5 个参数，第 1 个和第 2 个分别为计算步长和终止时间，第 3 个为起始仿真时间，第 4 个最大允许步长，最后的 UIC 表明需要用户指定初始条件。这时可以将其复制到 MATLAB 的 work 目录下，这样就可以给出如下的命令

```
>> prg='c:\spice\spice3.exe '; matd='c:\matlab6p1\work\c5ftri1.';
   aout=spice([prg matd 'cir -o ' matd 'out -b -m'],[],[matd,'log']);
```

命令 spice.dll 自动调用 Spice 程序，对描述的电路进行仿真，将结果返回到 MATLAB 工作空间的 aout 变量，该变量的第一列为时间向量，第二列和其他可能的后续列为原系统要求输出的信号，在本例中为节点 1 的电压。这样，在 MATLAB 提示符下键入

```
>> plot(aout(:,1),aout(:,2))
```

就可以绘制出仿真的结果，如图 5-99 (b) 所示。

假设想用一组新的信号来取代原模型的 6V 直流电压

```
>> tt=aout(:,1); yy=5+sin(1e7*tt);
   plot(tt,yy), set(gca,'ylim',[0,6]); i_sig=[tt yy]';
```

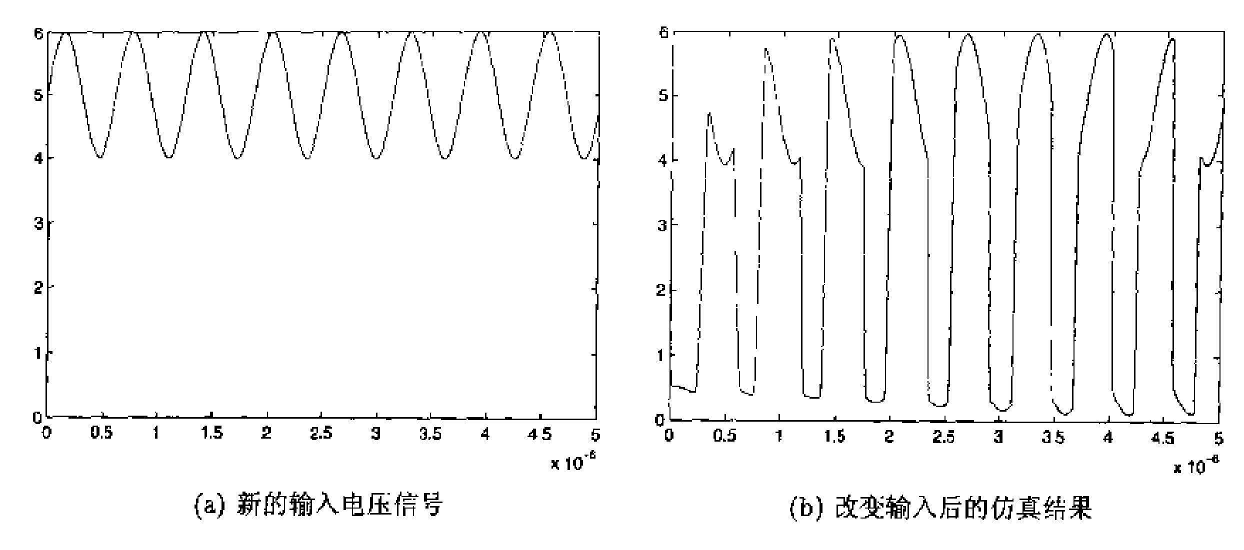
则可以得出如图 5-100 (a) 所示的输入电压波形，可以将时间变量 tt 和电压变量 yy 作为输入信号的两行，构成 i_sig 变量，可以用这个变量来驱动 Spice 模型。具体的方法是：将电压源输入语句

```
V0 3 0 DC 6V
```

改变为

```
V0 3 0 matlab 1  * 电压模块，从节点 3 到 0
```

并将其存入 c5ftri2.cir 文件，则可以用下面的语句对之进行仿真



(a) 新的输入电压信号　　(b) 改变输入后的仿真结果

图 5-100 电路仿真结果

```
>> prg='c:\spice\spice3.exe '; matd='c:\matlab6p1\work\c5ftri2.';
   aout=spice([prg matd 'cir -o ' matd 'out -b -m'],i_sig,[matd,'log']);
   plot(aout(1,:),aout(2,:))
```

并将结果绘制出来，如图 5-100 (b) 所示。

由这个例子可以看出，可以在 MATLAB 下对整个系统进行仿真。就是说可以由预先设定的信号取代原来的电源信号，从而对整个系统进行仿真。遗憾的是，不能将其嵌入到 Simulink 模型中进行仿真，即令其输入信号为动态计算出的信号，所以在系统仿真中还是有限制的。

5.5 非线性系统控制设计模块集

非线性系统控制设计 (Nonlinear Control Design，简称为 NCD) 模块集是一个基于最优化技术进行系统时域设计的实用工具。打开该模块集可以得出如图 5-101 所示的模块，

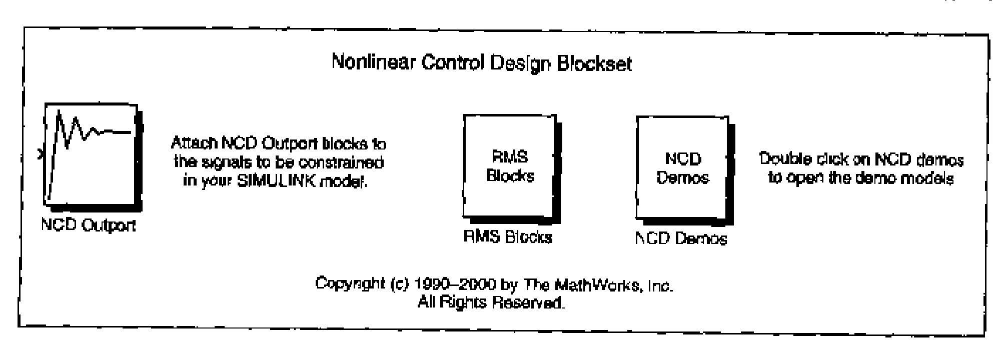


图 5-101 非线性系统控制设计模块集

用户可以构造出控制系统的 Simulink 框图，NCD Outport 可以接到系统的输出端口上，然后利用 NCD 模块提供的功能对控制系统内某些参数进行优化。NCD 模块还具有某些

鲁棒控制设计的功能，假如系统中含有不确定性，用 NCD 模块能够设计出鲁棒的控制器。

〖例 5.25〗 考虑一个 PID 控制系统的例子，假设系统的对象模型为 $G(s)=10e^{-s}/(s^2+a_1s+a_2)$，并假设 $a_1=0.1, a_2=4$。在实际控制中，不允许控制信号过大，所以经常存在一个驱动限幅非线性环节，并设置饱和限幅值为 c_1。假设使用 PID 控制器，且其初值 $K_p=5, K_i=5, K_d=1$。可以建立起如图 5-102 所示的框图，并设置终止仿真时间为 4。在该框图中在系统输出口上附加了一

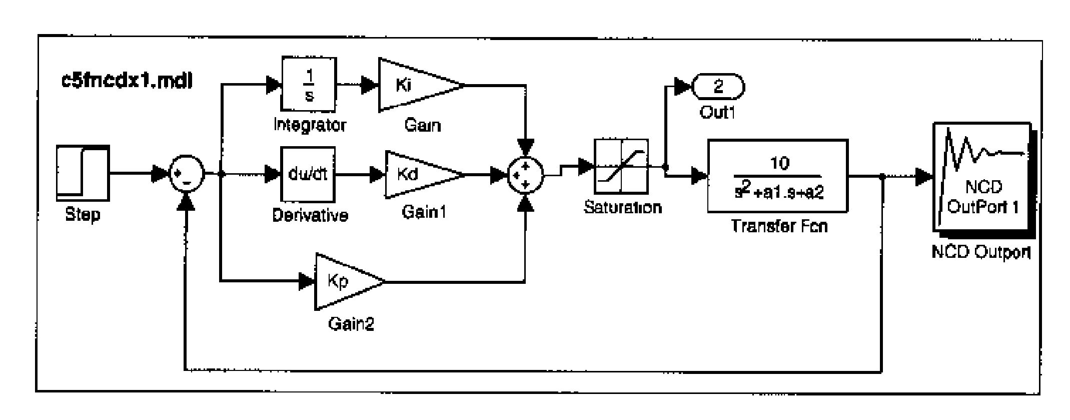


图 5-102 非线性系统的 PID 控制器设计框图

个 NCD Outport 模块。可以用下面的语句设置各个参数的初值

```
>> a1=0.1; a2=5; Kp=10; Ki=1; Kd=6; c1=2;
```

双击 NCD 模块，则得出如图 5-104 所示的界面，在该对话框中允许用户用图形的方式选择系统阶跃响应的各个指标，如超调量、上升时间和调节时间等，调整方式很简单，只需拖动水平或垂直滚动杆就可以设置期望的响应区域。

如果想获得最优的 PID 控制器参数，则需要选择其 Optimization | Parameters 菜单项，这将得出如图 5-104 所示的对话框，在其中的 Tunable parameters (可调参数列表) 栏目中填写待定的参数 Kp, Ki 和 Kd，同时将这些变量的最小值都设置为 0，按下 Done 按钮关闭该对话框，可以单击约束设置界面中的 Start 按钮，就可以寻优并动态的显示曲线结果，如图 5-105 所示。在该图形中显示两条曲线，效果差的是初始响应效果，经过寻优过程，将得出较满意的响应曲线。同时在命令窗口中显示如下结果：

```
Processing uncertainty information.
Uncertainty turned off.
Setting up call to optimization routine.
Done plotting the initial response.
Start time: 0    Stop time: 4.
There are 162 constraints to be met in each simulation.
There are 3 tunable variables.
There are 1 simulations per cost function call.
Creating a temporary SL model tp064152 for computing gradients...
Creating simulink model tp064152 for gradients...Done
```

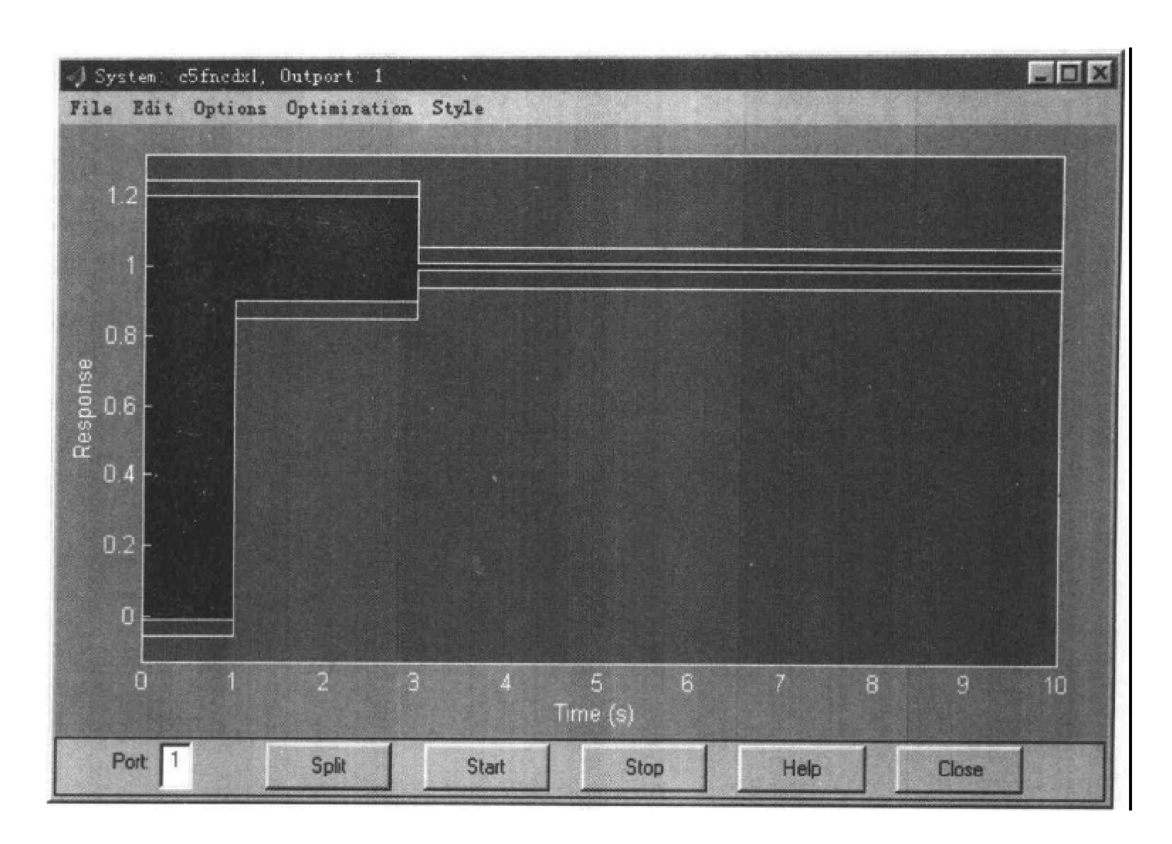


图 5-103 阶跃响应约束设置界面

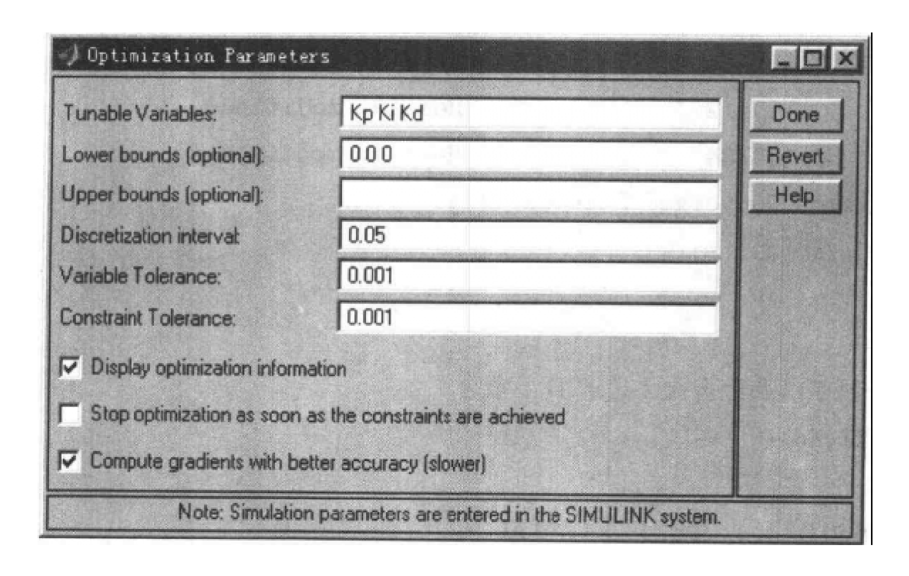


图 5-104 参数设置对话框

```
f-COUNT     MAX{g}        STEP  Procedures
      7    0.122591         1
     14    0.122521         1   Hessian modified twice
     23    0.122596      0.25   Hessian modified
     30    0.122663         1   Hessian modified
```

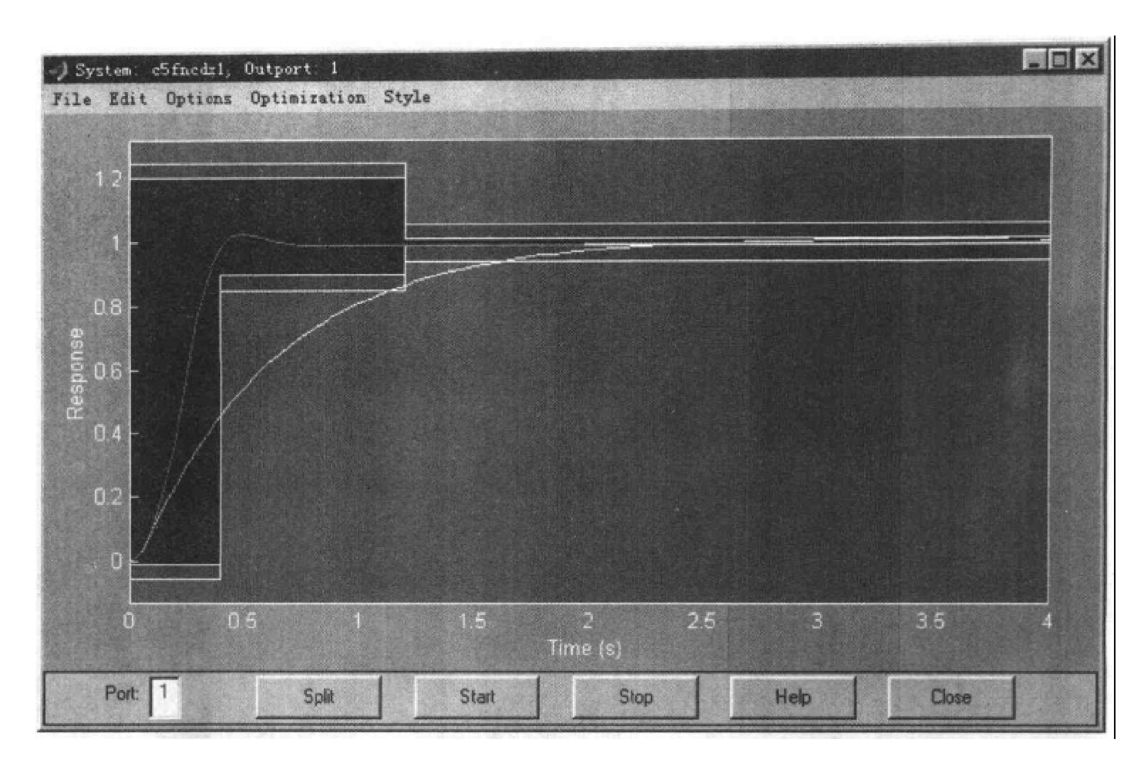


图 5-105　在当前约束下的寻优效果

```
    37      0.122229           1    Hessian modified twice
    44      0.111414           1
    51      0.110812           1    Hessian modified
    52      0.110784           1    Hessian modified
Optimization Converged Successfully
Active Constraints:
     10
     88
```

给出如下的命令则可以显示出最优的 PID 参数

```
>> [Kp,Ki,Kd]
ans =
    12.6507    1.7952    1.4950
```

还可以通过调整约束区域的方法更改设置，再进行寻优过程，则最终将得到如图 5-106 所示的结果。得出的 PID 参数为:

```
>> [Kp,Ki,Kd]
ans =
    4.2530    1.3796    0.7845
```

当然，这样设置控制约束的方法并不能无限制地进行下去，毕竟某些指标是实际系统所无法达到的，所以应该合理地选择指标，可以采用试凑的方式来寻找这样合理的指标。

该模块集还支持含有不确定参数系统的控制器设计，例如若选择 Optimization | Uncertainty

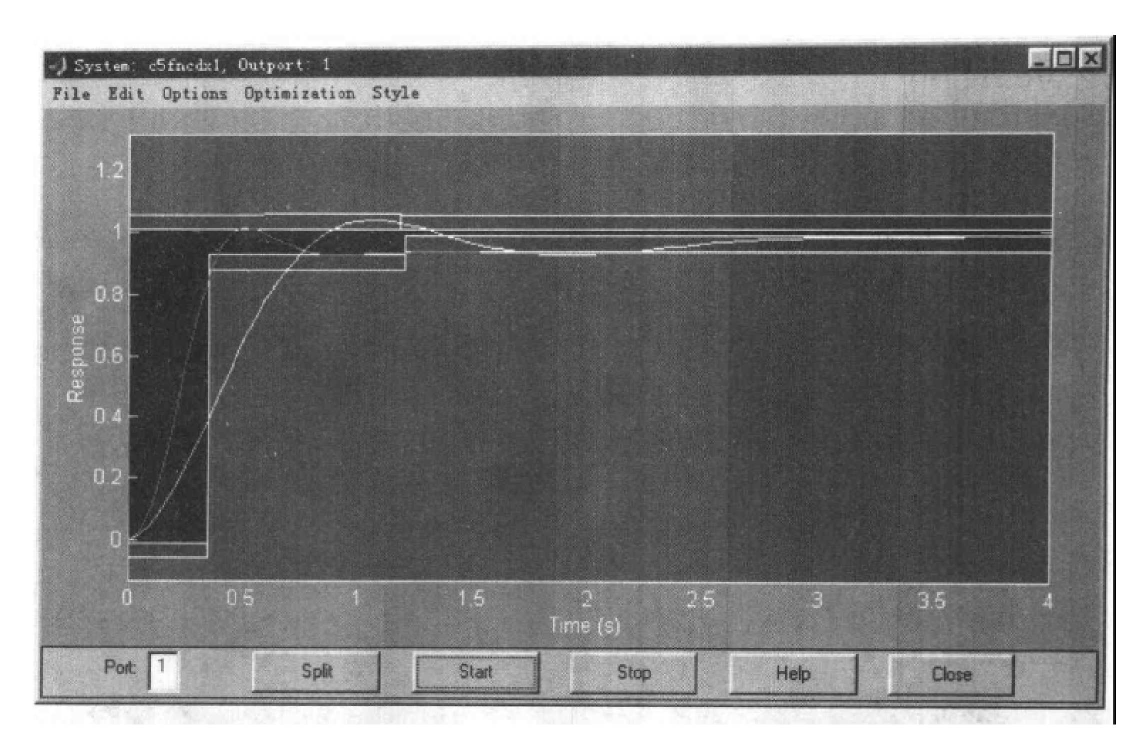


图 5-106 改进的控制效果

菜单项，则将得出如图 5-107 所示的对话框，在其中设定不确定参数的范围，如让 $a_1 \in [0.01, 1]$，

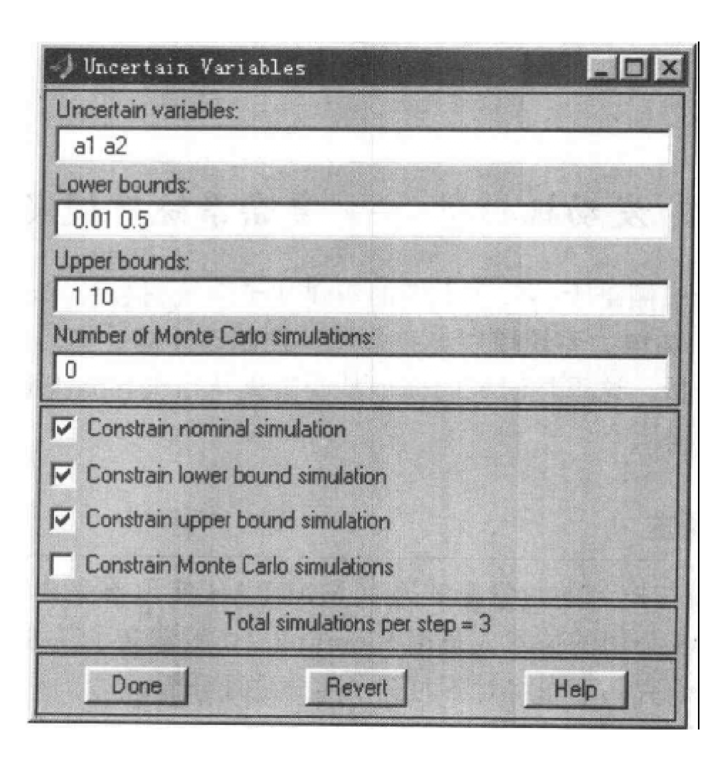


图 5-107 不确定参数设置对话框

$a_2 \in [0.5, 10]$ 的范围内变化，并选中约束上下界，选中显示不确定系统响应的上下界，再启动寻优过程，则将得出如图 5-108 所示的控制效果，这时的控制器参数为

```
>> [Kp Ki Kd]
ans =
   12.9933    1.9998    1.6846
```

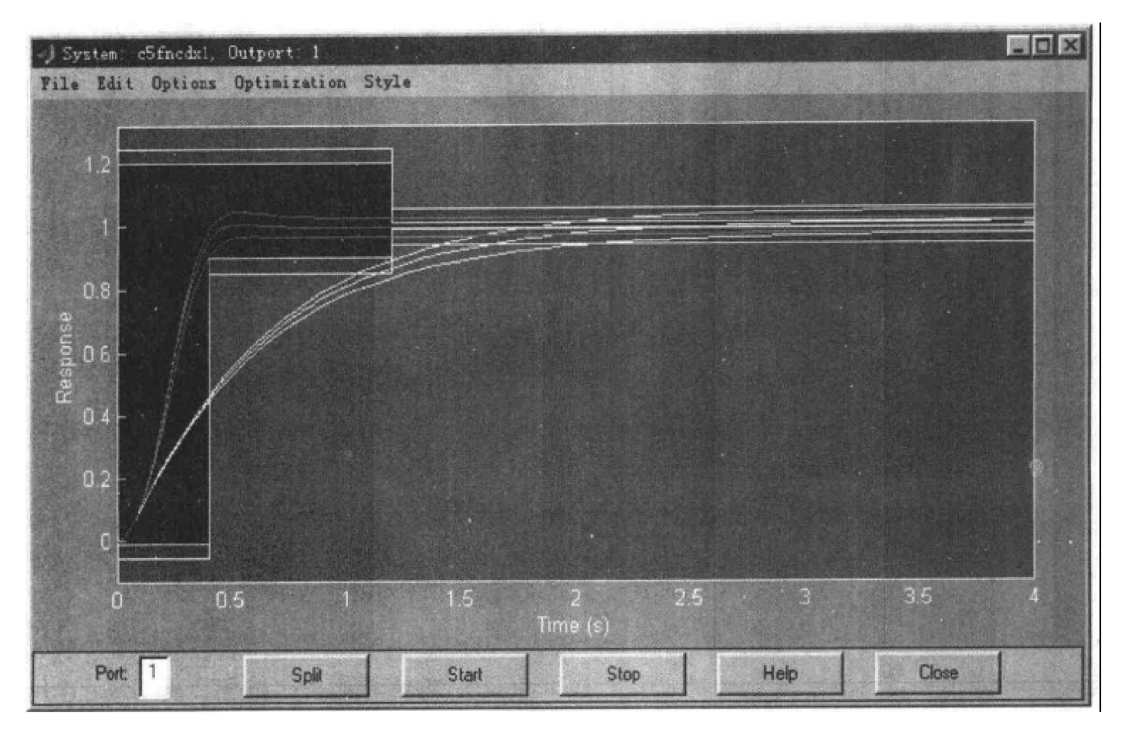


图 5-108 新的 PID 控制器效果

5.6 发动机模型——复杂系统建模实例

本章前面的部分中侧重于一般子系统的搭建和子系统封装技术，并介绍了一些实用的 Simulink 模块集，如电力系统模块集和非线性控制设计模块集，本节中对一个实际的复杂过程进行个案分析，这里将研究的模型是四缸火花点火 (spark ignition) 发动机模型的建立与仿真的全过程。

5.6.1 模型背景概述

四缸内燃机建模与控制的数学形式及其 MATLAB 实现首先由 Crossley 和 Cook 提出[7], The MathWorks 公司的技术人员用 Simulink 2.0 版建立了其 Simulink 模型[32]，并对其控制进行了仿真研究。下面将对下面五个部分进行分析。

(1) 活门 (throttle)

(2) 进气导管 (intake manifold)

(3) 质量流速 (mass flow rate)

(4) 转矩产生与加速 (torque generation and accelaration)

(5) 压缩冲程 (compression stroke)

事实上，为获得更精确的仿真结果，还可以在上面加上其他的部分，这完全取决于原始问题的物理建模。

5.6.2 发动机模型分析

5.6.2.1 活门模型

在仿真分析中首先需要研究活门部分，在该部分中控制输入信号为活门的角度 θ，单位为度。

进气导管导入空气速度的模型可以由两个函数的积来表示

$$\dot{m}_{ai} = f(\theta)g(P_m) \tag{5.5}$$

式中 $\dot{m}_{ai}$ 为流入进气导管的质量流速，P_m 为导管压力。该乘积中一个因子是关于活门角度 θ 的经验公式函数 $f(\theta)$，它可以写成

$$f(\theta) = 2.821 - 0.05231\theta + 0.10299\theta^2 - 0.00063\theta^3$$

另一个因子是导管压力的函数 $g(P_m)$

$$g(P_m) = \begin{cases} 1, & P_m \leq P_a/2 \\ 2\sqrt{P_m P_a - P_m^2}/P_a, & P_a \geq P_m > P_a/2 \\ -2\sqrt{P_m P_a - P_a^2}/P_a, & P_a \leq P_m \leq 2P_a \\ -1, & P_m \geq 2P_a \end{cases} \tag{5.6}$$

其中 P_a 为大气压。可以将该式简化成

$$g(P_m) = \begin{cases} 2\sqrt{P_m P_a - P_m^2}\text{sign}(P_a - P_m)/P_a, & 2P_a \geq P_m > P_a/2 \\ \text{sign}(P_a - P_m), & \text{其他情况} \end{cases} \tag{5.7}$$

令活门角度 (theta)、导管压力 (Pm) 和大气压 (Pa) 为输入信号，流入进气导管的质量流速 (dx_ai) 为输出信号，则可以构造出子系统模型，如图 5-109 所示，其中开关元件的阈值设置为 0.5。在该模型中用了一种巧妙的方法来描述式 (5.7) 中的逻辑式 $2P_a \geq P_m > P_a/2$。

5.6.2.2 进气导管

进气导管的仿真模型可以表示成导管压力的微分方程

$$\dot{P}_m = \frac{RT}{V_m}(\dot{m}_{ai} - \dot{m}_{ao}) \tag{5.8}$$

其中 R 为给定的常数，T 为温度 (单位为 °K), V_m 为导管容积 (单位为 m^3), $\dot{m}_{ao}$ 为导管进气口空气的质量流速 (单位为 g/s), $\dot{P}_m$ 为导管压力的变化率 (单位为 bar/s)，在该系统中，$RT/V_m = 0.41328$。

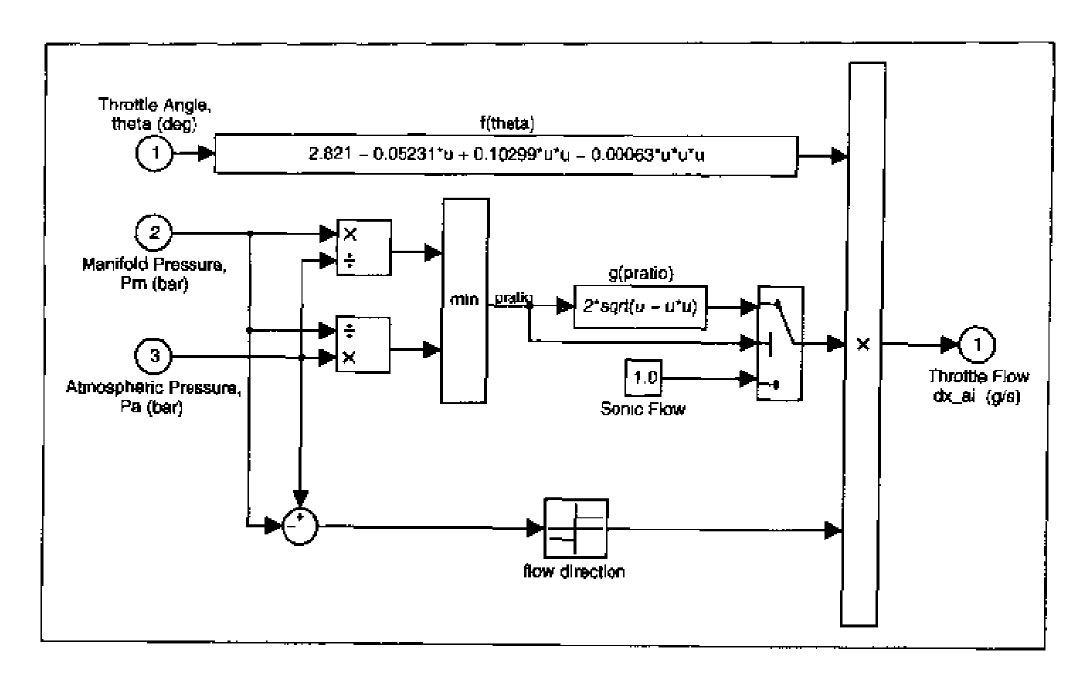


图 5-109　活门导管模型子系统

导管进气口空气的质量流速可以由下面的经验公式表示

$$\dot{m}_{ao} = -0.366 + 0.08979NP_m - 0.0337NP_m^2 + 0.0001N^2P_m \tag{5.9}$$

式中 N 为发动机的速度 (单位为 rad/s)。

以 $\dot{m}_{ai}$ 和 N 为输入信号，以导管进气口空气的质量流速 $\dot{m}_{ao}$ 和导管压力 P_m 为输出信号，可以绘制出它们之间关系的子系统模型，如图 5-110 所示。

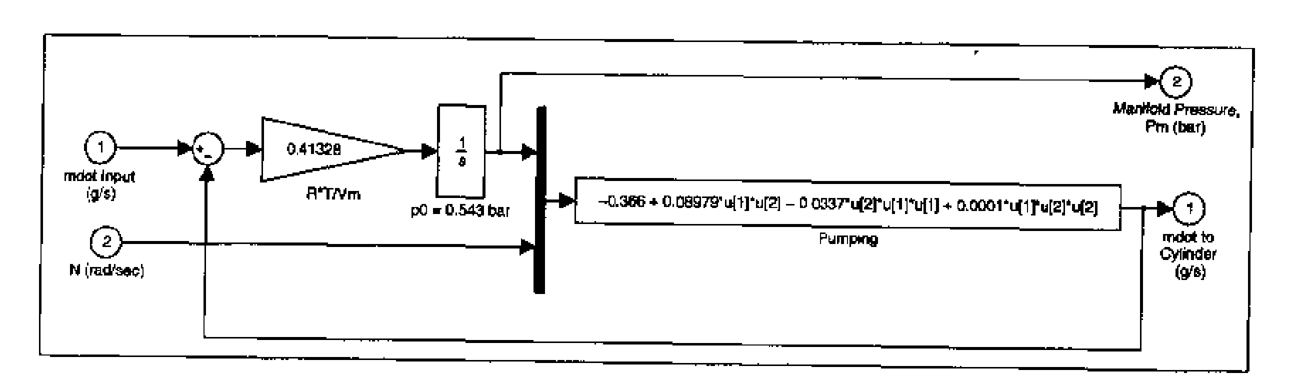


图 5-110　进气导管模型子系统

假设大气压 $P_a = 1$ bar/s，综合考虑这两个子系统，可以构造出活门角度 θ 和发动机速度 N 为输入信号，以进气口空气的质量流速 $\dot{x}_{ao}$ 为输出信号，可以构造出活门和导管部分总的子系统，如图 5-111 所示。

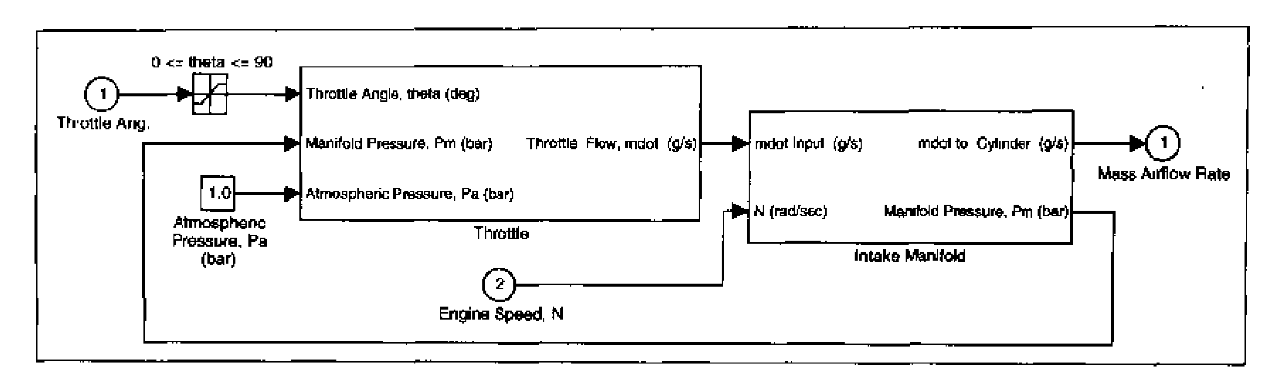


图 5-111 活门导管模型子系统

5.6.2.3 转矩产生与加速模型

现在讨论发动机产生的转矩模型，该模型由下面的经验公式表示

$$
\begin{aligned}
T_{\text{eng}} = &-181.3 + 379.36m_a + 21.91(A/F) - 0.85(A/F)^2 + 0.26\sigma - 0.0028\sigma^2 \\
&+0.027N - 0.000107N^2 + 0.00048N\sigma + 2.55\sigma m_a - 0.05\sigma^2 m_a
\end{aligned}
\tag{5.10}
$$

式中 m_a 为汽缸内气体的质量 (单位为 g), A/F 为空气和燃料的比率，在本例中取 1/14.6, σ 为点火提前量 (单位为角度，在本系统中取 15°), T_{eng} 为发动机产生的转矩 (单位为 Nm)。根据上面的公式，可以绘制出转矩的模型，如图 5-112 所示。

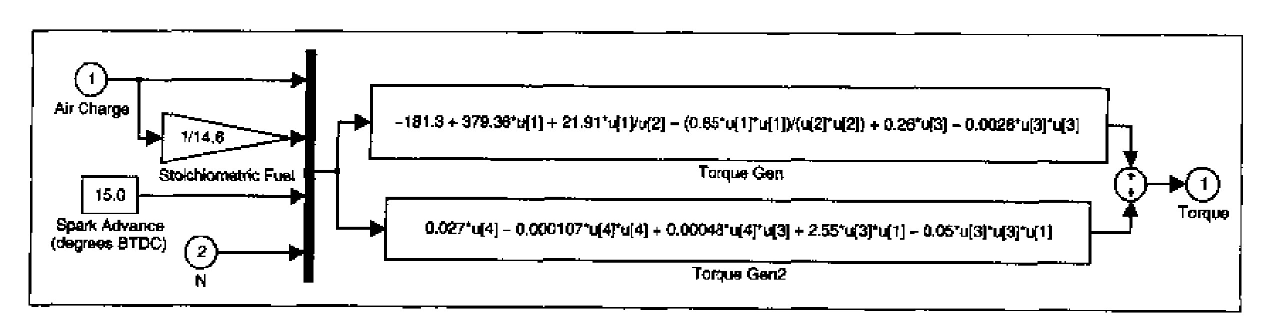


图 5-112 燃烧时转矩产生子系统

由发动机的转矩减去负载转矩，就可以计算出发动机的加速度

$$
\dot{N} = \frac{1}{J}(T_{\text{eng}} - T_{\text{load}})
\tag{5.11}
$$

式中 J 为发动机的转动惯量 (单位为 kg-m^2)，在本例中可以取为 0.14 kg-m^2, T_{load} 为负载转矩 (单位为 Nm)，$\dot{N}$ 为发动机的加速度 (单位为 rad/s^2)。可以根据该公式搭建出 Simulink 子系统模型，如图 5-113 所示，在此子系统中，设置转速积分器的初值为 209.48 rad/s，表明只研究发动机转速处于稳态时系统的性能。可以看出，此子系统是很简单的。

5.6.2.4 压缩冲程

在四缸四冲程 (进气、压缩、燃烧和排气) 的发动机内，曲柄轴每旋转 180° 将改变

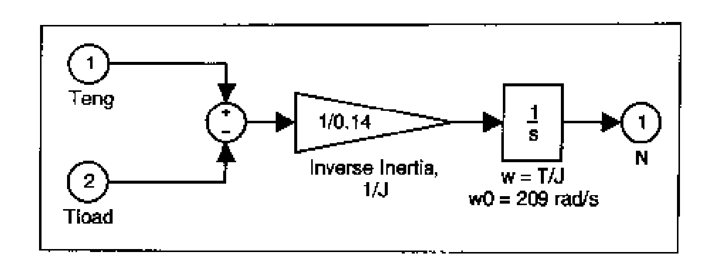
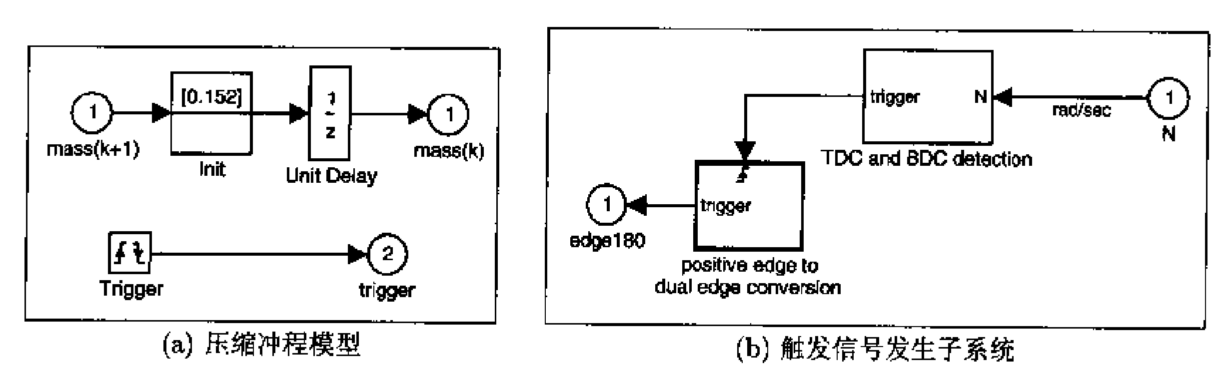


图 5-113　燃烧时转矩产生子系统

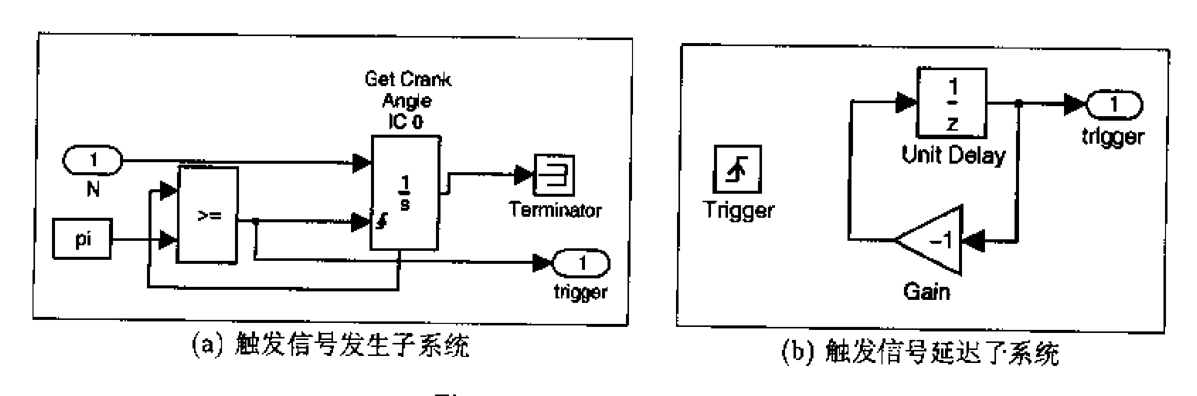
相邻汽缸的点火状态，这将使得每隔一次曲柄轴转动周期，每个汽缸才能点火一次。这样，相对进气冲程来说，压缩冲程、燃烧冲程将依此延迟 180°。所以需要采用触发子系统的方式来描述压缩冲程，其子系统如图 5-114 (a) 所示，其触发信号发生子系统如图 5-114 (b) 所示。



(a) 压缩冲程模型　　(b) 触发信号发生子系统

图 5-114　压缩冲程与触发信号模型

在触发子系统中，有两个下级子系统，分别如图 5-115 (a) 和 (b) 所示，它们分别用来描述触发信号的发生和延迟。



(a) 触发信号发生子系统　　(b) 触发信号延迟子系统

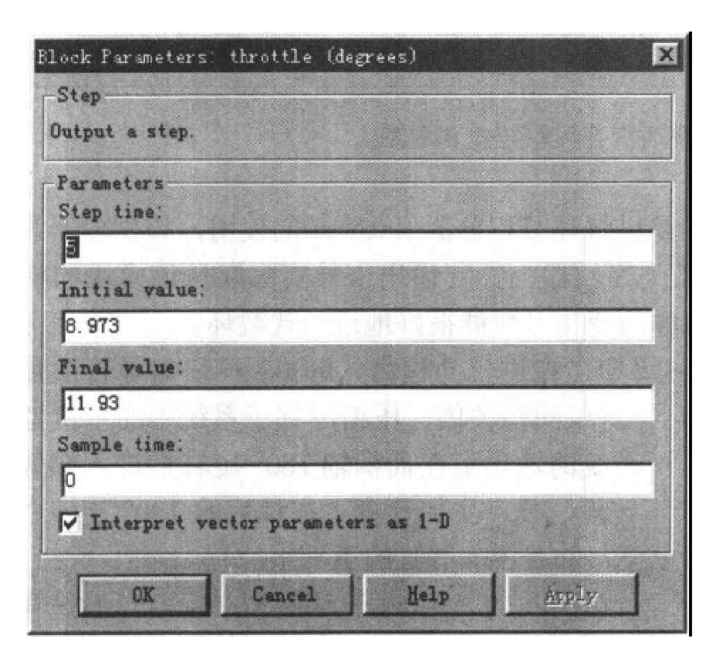
图 5-115　触发信号子系统

5.6.2.5 输入角度设定和负载转矩

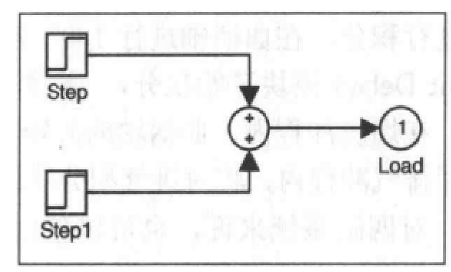
活门角度假设由下式设定

$$\theta=\begin{cases}8.97, & t<5\\ 11.93, & t\geqslant 5\end{cases}$$

根据该式，可知该输入可以由单个的阶跃输入环节构造出来，双击该环节可以得出该模块的参数对话框，可以按照图 5-116 (a) 的格式填写即可。



(a) 活门角度阶跃信号对话框



(b) 负载输入子系统

图 5-116 发动机模型输入信号描述

还可以假设负载信号由下式描述

$$T_{\rm load}=\begin{cases}25, & t<2\\ 20, & 2\leqslant t<8\\ 25, & t\geqslant 8\end{cases}$$

可以按图 5-116 (b) 所示的方式搭建负载转矩设置子系统，并用类似于活门角度的方法设计出两个阶跃模块的参数。

5.6.3 开环系统的建模与仿真

建立了这些子系统后，可以搭建出整个发动机开环系统模型，如图 5-117 所示。在该系统模型中，设置了两个示波器，其一用于显示发动机的转速，另一个用于显示设定点输入信号 θ 和负载转矩 $T_{\rm load}$。

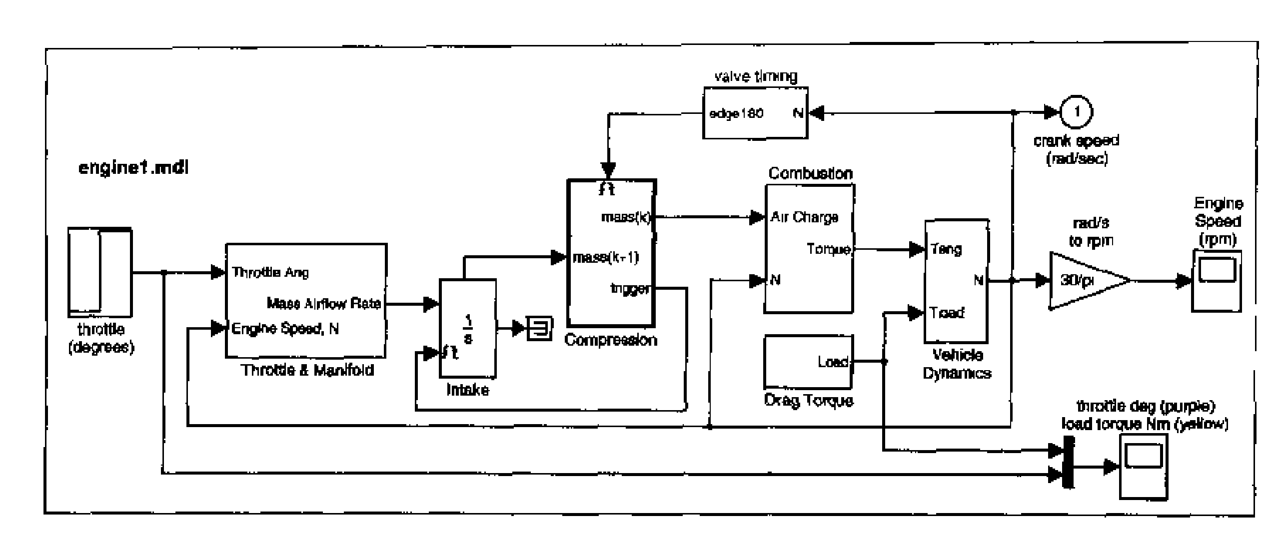


图 5-117　发动机开环系统仿真框图

在该系统模型中另一点值得说明的是进气环节积分器 (Intake) 的使用，该积分器为了避免代数环的产生，其输出端直接连接到终结器上，其输出信号实际取自该模块的状态输出端。在积分器带有触发信号时，这样的变化一般能很好地避免代数环。

现在考虑一个汽缸的完整的四冲程周期，在进气冲程内，Intake 模块对导管质量流速进行积分，在曲柄轴进行 180° 旋转后，进气阀门关闭，压缩冲程子系统中的单位延迟 (Unit Delay) 模块开始积分， 压缩冲程子系统的输出值在曲柄轴 180° 旋转后进入燃烧冲程，在燃烧冲程内，曲柄轴的旋转将产生转矩，并产生加速度。在最后的 180° 旋转中，亦即排气冲程内，将对进气积分器进行复位，并准备本汽缸的下一个冲程。

对四缸系统来说，也可以使用 4 个进气模块、4 个压缩子系统等来建模，但这样处理后每个子系统将空闲 75% 的时间，所以在建模时用这样更有效的方式来实现原始的过程。

可以按图 5-118 (a) 中所示的方式设置示波器的结果输出方式，亦即将示波器仿真的结果显示出来的同时，还将结果按带有时间的结构体形式传入 MATLAB 的工作空间中，形成变量 N。仿真结束后，可以给出如下的命令绘制发动机速度输出曲线，如图 5-118 (b) 所示。

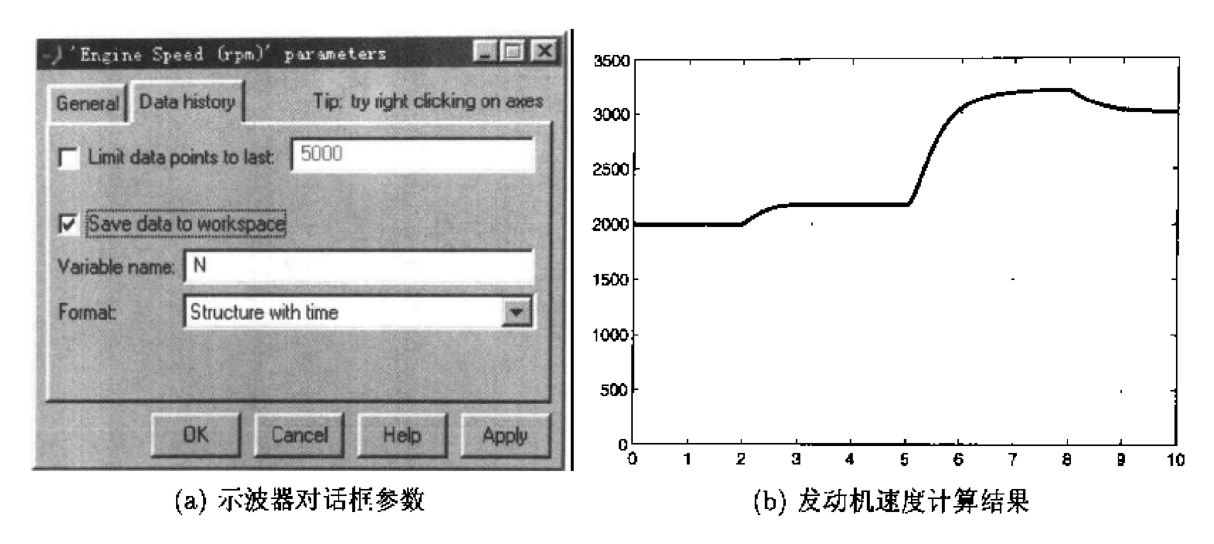
```
>> plot(N.time,N.signals.values) % 绘制曲线
   set(gca,'Ylim',[0,3500]), grid  % 设置 y 轴范围并给图形加网格修饰
```

显然，当负载转矩发生变化时，发动机速度变化很大，这就是开环控制的弊病，所以有必要为之设计控制器，使得在闭环控制下得到更好的控制效果。

5.6.4　闭环系统的建模与仿真

可以选择比例加积分 (PI) 控制的策略设计控制器，其中积分控制是为了在工作点发生变化时消除静态误差而引入的，PI 控制器的模型为

$$\theta(t)=K_p(N_{\rm set}-N(t))+K_I\int_0^t(N_{\rm set}-N(\tau))\mathrm{d}\tau \tag{5.12}$$



(a) 示波器对话框参数　　(b) 发动机速度计算结果

图 5-118　发动机速度设置与结果

式中 N_{set} 为速度设定点的值，K_p 为 PI 控制器中的比例增益，K_I 为积分器的增益。这样可以构造出如图 5-119 所示的数字 PI 控制器的框图。

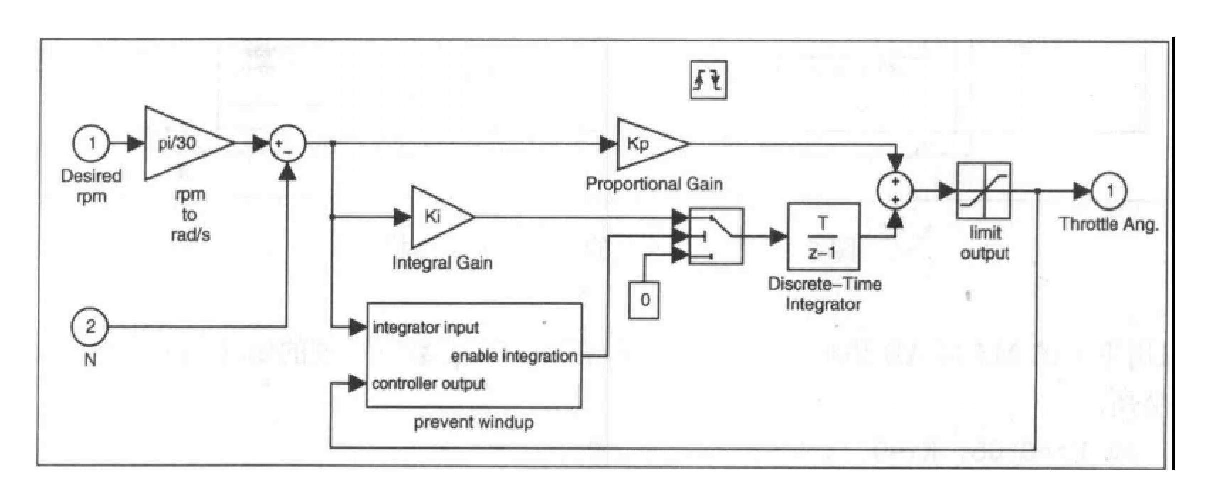


图 5-119　数字 PI 控制器子系统模型

可以看出，这样的 PI 控制器和式 (5.12) 中的是不一致的，首先其积分部分是数字式的，而不是连续式的；此外，由于控制器的执行必须与发动机曲柄轴的转动同步，所以应该由触发子系统来表示；另外，实际 PI 控制的结构中需要对积分器的回绕做出抑制，所以这里引入了一个如图 5-120 所示的子系统，在回绕出现时能发出信号，使得图 5-119 子系统中的积分器清零，达到抑制回绕的作用。

将这样设计出来的控制器加入原始的开环系统模型，就能立即得出如图 5-121 所示的闭环控制系统模型。可以看出，该模型类似于开环模型，所不同的是在前端加了一个改进的 PI 控制器。

如果受控对象模型和前面开环系统完全一致，且设定点与负载转矩也完全一致，则

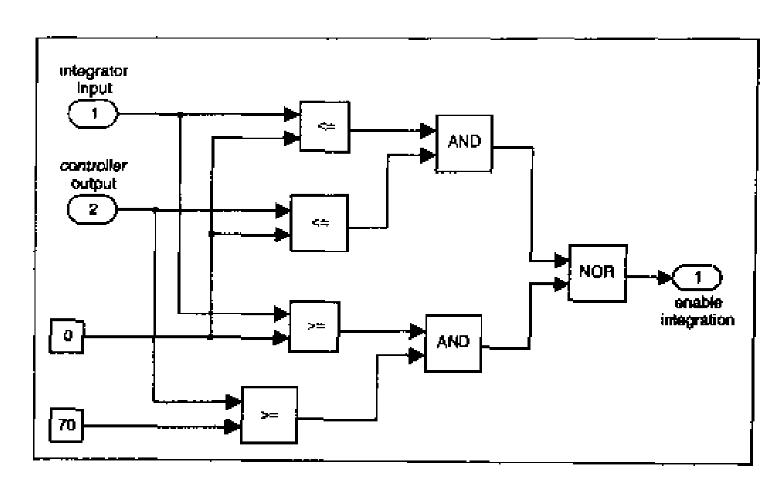


图 5-120 PI 控制器中的抗饱和子系统

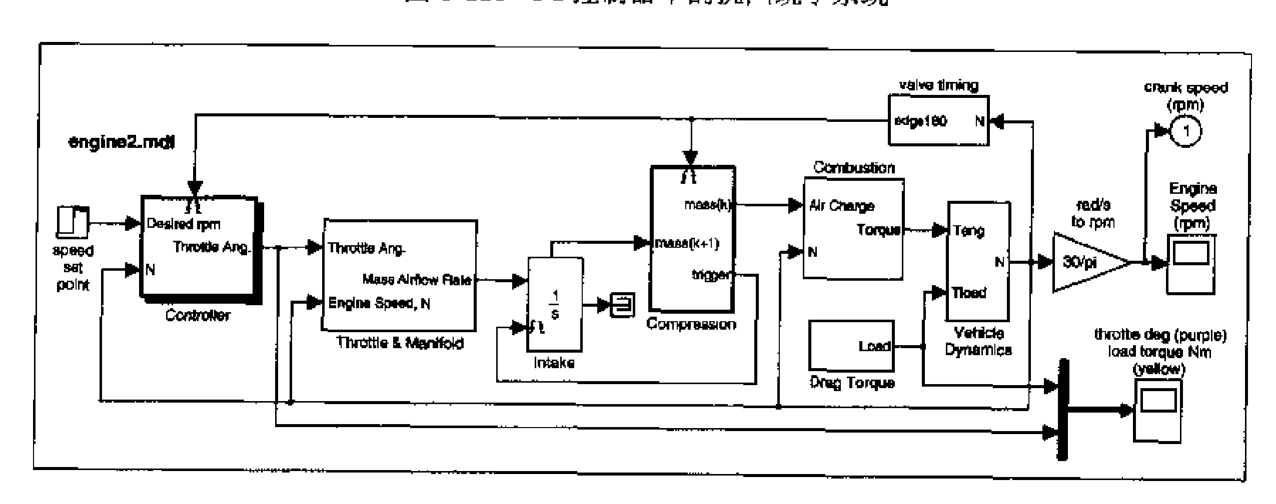


图 5-121 速度调节的闭环控制系统模型

可以用下面的 MATLAB 语句[1] 计算并绘制出三组 PI 参数下系统的闭环响应曲线，如图 5-122 所示。

```
>> Kp=0.05; Ki=0.1; % 第一组 PI 参数
   [t1,x1,y1]=sim('engine2'); % 仿真系统，得出时间、状态和输出
   Kp=0.033; Ki=0.064; % 第二组 PI 参数
   [t2,x2,y2]=sim('engine2');
   Kp=0.061; Ki=0.072; % 第三组 PI 参数
   [t3,x3,y3]=sim('engine2');
   plot(t1,y1,'-',t2,y2,'--',t3,y3,':')
   grid; set(gca,'Ylim',[0,3500])
```

可以看出，在闭环控制下，发动机转速受负载转矩扰动显著减小，且在较短的时间内消除影响，所以在实际控制中多采用闭环形式。

前面的仿真分析都是在系统有初始状态的假设基础上进行的，亦即假设系统已经进

[1] 有关 sim() 函数的格式和其他功能将在第 6 章中详细介绍。

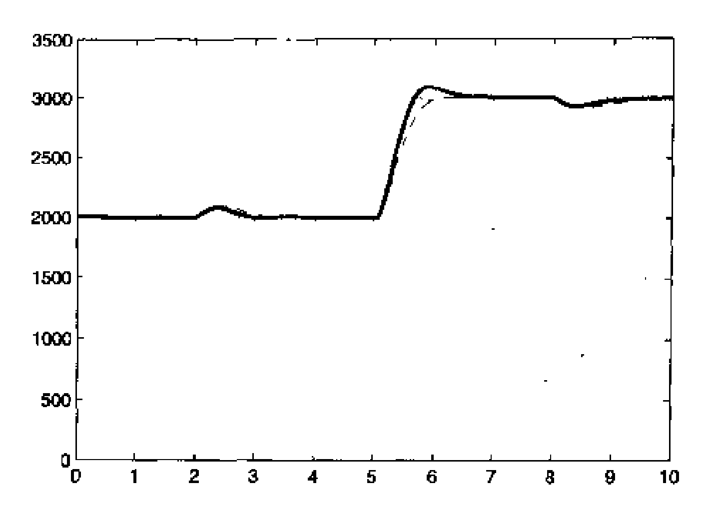


图 5-122 稳态闭环系统仿真结果

入稳态，然后才分析系统在扰动下的抗扰性能，打开 Vehicle Dynamics 子系统，可以发现其积分器是有初始值的。现在讨论包括启动过程在内的全过程仿真。首先应该将该积分器的初值设置为 0，此外对各个设定点和转矩扰动信号如表 5-2 进行调整，得出新的模型 engine3.mdl。给出下面的 MATLAB 命令，

```
>> [t,x,y]=sim('engine3',40); % 终止时间设置为 40
```

则进行仿真将得出如图 5-123 所示的响应曲线。可以看出，这里的控制算法仍然适用于全过程的仿真。

表 5-2 输入模块参数修改表

模块名称	阶跃发生时刻 Step time	初始值 Initial value	终止值 Final value
设定点模块	15	2000	3000
转矩扰动 Step	20	0	20
转矩扰动 Step1	30	0	5

5.7 习 题

(1) 用向量输入积分器的方法表示 Lorenz 方程的 Simulink 模型，并进行仿真运算，比较和原来 Simulink 模型的差异。

(2) 假设已知误差信号 $e(t)$，试构造出求取 ITAE, ISE, ISTE 准则的封装模块。要求：误差信号 $e(t)$ 为该模块的输入信号，双击该模块弹出一个对话框，允许用户用列表框的方式选择输出信号形式，将选定的 ITAE, ISE, ISTE 之一作为模块的输出端显示出来。其中 ISTE 的定义为

$$J_{\rm ISTE}=\int_0^t \tau e^2(\tau){\rm d}\tau$$

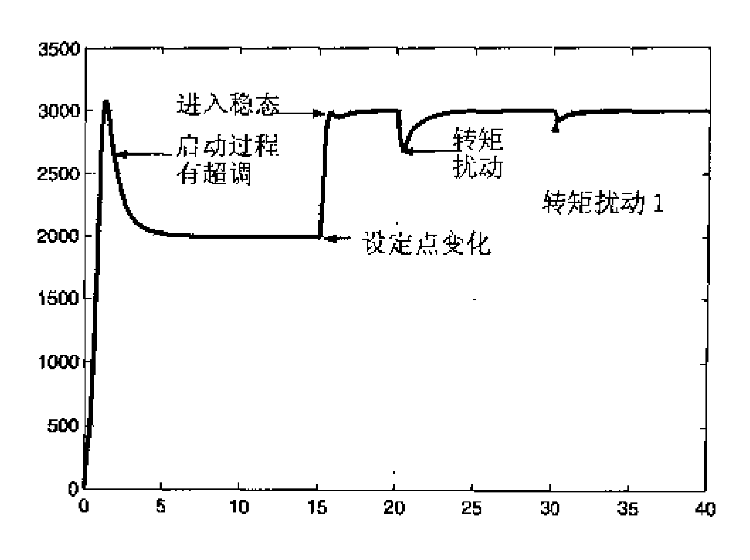


图 5-123　包括启动过程的闭环系统仿真结果

(3) 考虑 Lorenz 方程模型，该模型没有输入信号，假设选择其三个状态变量 $x_i(t)$ 为其输出信号，以 β,σ,ρ 和 $x_i(0)$ 向量为附加参数，试将该模块封装起来，并绘制在不同参数下的 Lorenz 方程解的三维曲线。

(4) 利用模块封装技术，由给定的 RLC 串联模块拆分出单个的电阻、电感和电容元件，并将其写入一个新的模块组。

(5) 考虑如图 5-124 所示的电路，假设已知 $R_1=R_2=R_3=R_4=10\Omega, C_1=C_2=C_3=10\mu\mathrm{F}$，并假设输入交流电压 $v(t)=\sin(\omega t)$，并令 $\omega=10$，试对该系统进行仿真，求出输出信号 $v_c(t)$，并求出其解析解。可以用两种方法绘制出该系统的 Bode 图：① 求出系统的线性模型，然后用 bode() 函数绘制，② 选择一系列不同的频率值 ω_i，在每一个频率值下求出解析解的幅值与相位，然后用描点的方法绘制出 Bode 图，请问二者是否一致。

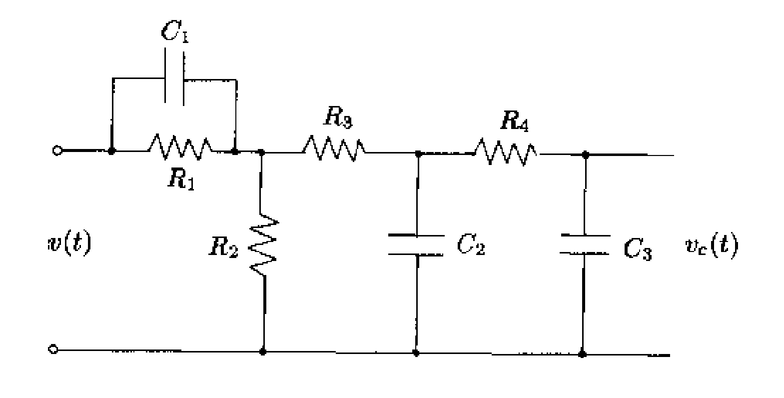


图 5-124　习题 (5) 图

(6) 三极管是电子线路中经常使用的元件，而电力系统模块集没有提供该元件，给电子线路的仿真带来很大的困难。在电子线路类课程中通常介绍三极管的近似方法，试利用近似方法用

Simulink 模块搭建出三极管的等效电路，并封装成可用模块。

考虑如图 5-125 所示的逆变器结构，其目的是将输入的直流电压 E 经过逆变器输出接近于正弦的三相电压 a,b,c，产生三相电，需要按 T_1 到 T_6 的顺序设计触发器，使得每个触发脉冲的宽度为 120°，即总脉冲宽度的 1/3。另外应该假设触发器脉冲发生时刻按 $T_1, T_2, \cdots, T_5, T_6$ 的顺序每个延迟 60°，试对这样的系统进行仿真，并观测得出的三相点电压波形。

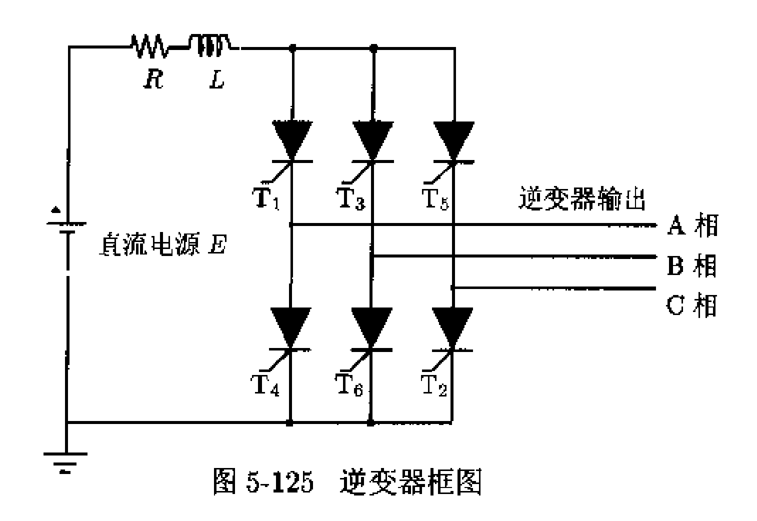


图 5-125 逆变器框图

(8) 给定受控对象模型为

$$G(s) = \frac{100}{s(s+10)(s+20)(s+30)}e^{-10s}$$

试利用非线性设计模块集为其设计最优的比例加微分 (即 P-D) 控制器，并观察控制效果。如果受控对象模型发生变化，例如时间延迟常数变化为 8，用原来设计的控制器是否仍能较好地控制该过程？

(9) 在前面介绍的四缸四冲程发动机的闭环控制系统中，如果在仿真前已进入稳定区域，试通过仿真方法找出最大的容许负载值。

(10) 考虑四缸发动机的动态过程，利用仿真方法说明如果在启动初期就存在较大的负载扰动，发动机能正常启动吗？

第 6 章 Simulink 仿真的高级技术

前面两章中以较大的篇幅介绍了 Simulink 的基本使用方法与技巧，本章将介绍利用 Simulink 进行系统仿真的高级技术。在 6.1 节中将介绍如何用 MATLAB 命令来绘制 Simulink 框图，利用该节提供的技术，可以用语句绘制出结构相似的系统模型。在第 6.2 节中将介绍如何用 MATLAB 命令进行模型仿真过程的调用、系统的线性化等内容。特别地，介绍了 Simulink 本身线性化功能的不足，建议出实用的解决方法，并将介绍纯时间延迟系统的 Padé 近似技术。通过前面演示的 Simulink 功能，不难发现，并非所有的数学模型都能利用 Simulink 模块轻易地搭建起来，有的模型可能更适合于 MATLAB 或其他语句描述，所以在第 6.3 节中，将介绍实现复杂功能的 Simulink 模块编写技术，例如 S-函数的编写方法，侧重于用 MATLAB 格式和 C 语言格式介绍编写 S-函数，并着重介绍 S-函数在自抗扰控制器系统仿真中的应用。第 6.4 节中将介绍应用于有限状态机下的 Stateflow 建模与仿真问题，该工具主要用于复杂监控过程建模与仿真方面，给 Simulink 本身建模提供了新的方法和策略，扩大了 Simulink 本身的功能。还将介绍在仿真流程中常用的条件模块和循环模块的 Simulink/Stateflow 实现。在第 6.5 节中，还将介绍输出结果的可视化方法，如虚拟现实技术及其应用。首先介绍虚拟现实建模语言 VRML 的编程基础，并通过例子演示虚拟现实技术的 MATLAB/ Simulink 实现。

6.1 Simulink 模型的语句修改

6.1.1 Simulink 模型与文件的处理

Simulink 文件的操作可以完全由模型窗口的 File 菜单中的相应子菜单实现。同样，模型的读写还允许用函数调用的形式完成。在 MATLAB 命令窗口中，调用 `new_system()` 函数，则可以在 MATLAB 的工作空间中建立一个空白的 Simulink 模型，但这个模型不能自动地显示出来，这样的模型在 Simulink 中又称为逻辑模型。`new_system()` 函数的调用格式为：

```
new_system(模型名，选项)
```

其中，模型名应该以字符串的显示给出，这样建立起来的逻辑模型一般就用该模型名表示了。而选项可以选择为 `'Library'` 和 `'Model'` 两种，前者将建立一个空白的模型库，后者将建立一个空白的模型，如果不给出第二输入变量，则建立起来的为逻辑模型。例如可以给出 `new_system('MyModel')` 命令来建立一个名为 `MyModel` 的逻辑模型。

建立了逻辑模型并不能直接显示出来，而需要给出 `open_system()` 函数来显示，该

函数的调用格式为：

```
open_system(模型名)
```

其中，模型名为一个字符串，它表示已经由 new_system() 函数建立起来的逻辑模型。例如，前面建立起来的逻辑模型可以由 open_system('MyModel') 命令直接打开，其实若只想打开已经创立的模型，则直接在 MATLAB 命令窗口中键入该模型名即可，例如键入 MyModel 命令。当然 open_system() 函数还有其他调用格式，后面将详细介绍。

Simulink 中提供了一系列处理模型的命令，如 find_system() 函数可以得出当前打开的全部 Simulink 模型的名称，如果打开多于一个模型，则将以单元数组的形式返回各个模型名的字符串。

模型建立起来之后，则可以使用 save_system() 函数将该模型存成模型文件，其后缀名为 mdl，该函数的调用格式为：

```
save_system(模型名, 选项)
```

如果不给出模型名，则将自动地存当前的模型，所以将该模型存成 mdl 文件，第二选项应该为文件名组成的字符串，如果给出第二选项，则将当前的模型存到一个新的文件中。

6.1.2 模型属性与模块属性

前面介绍过，用 new_system() 函数可以建立起一个空白的 Simulink 模型，这里将通过例子来给出空白模型的清单和简要介绍。

【例 6.1】 可以用下面的语句可以建立、打开一个空白的窗口，并存成一个 mdl 文件

```
>> new_system('newmodel'); % 建立逻辑模型
   open_system('newmodel');% 打开模型编辑窗口
   save_system('newmodel');% 将该模型存成 mdl 文件
```

这样可以将该模型存成 newmodel.mdl 文件[1]，其内容为

```
Model {                            % 模型模块引导语句
  Name              "newmodel"  % 模型名称
  Version           4.00        % Simulink 版本序号，事实上应该显示为 4.10
  SampleTimeColors      off     % 采样周期颜色标识
  LibraryLinkDisplay    "none"
  WideLines             off          % 是否需要粗线
  ShowLineDimensions    off          % 显示连线上信号路数
  ShowPortDataTypes     off          % 显示模块出口数据类型，从略
  ... ...
```

[1]用 MATLAB 6.1 建立起来的 Simulink 空白窗口 newmodel.mdl 在 MATLAB 6.0 下运行，则将得出一些警告信息，但一般情况下它们并不影响原模型的运行。

```
ToolBar          on              % Simulink 模型是否需要工具栏
StatusBar        on              % Simulink 模型是否需要状态栏
BrowserShowLibraryLinks off
BrowserLookUnderMasks   off
Created          "Wed Aug 08 07:47:29 2001" % 模型建立日期
UpdateHistory        "UpdateHistoryNever"   % 模型更新记录
ModifiedByFormat     "%<Auto>"              % 模型修改格式
ModifiedDateFormat       "%<Auto>"          % 模型修改时间的格式
LastModifiedDate     "Wed Aug 08 07:50:03 2001"   % 模型最后修改时间
ModelVersionFormat       "1.%<AutoIncrement: 0 >" % 模型版本号格式
ConfigurationManager     "None"
SimParamPage         "Solver"      % 以下为 Simulink 仿真的控制参数，可以由
StartTime        "0.0"             % Simulink 的 Parameters 对话框设定。
StopTime         "10.0"            % 起始和终止仿真时间
SolverMode           "Auto"        % 方程求解器选择
Solver           "ode45"           % 求解算法
RelTol           "1e-3"            % 相对误差容限
AbsTol           "auto"            % 绝对误差容限
Refine           "1"               % 中间插值点设置
MaxStep          "auto"            % 最大计算步长
MinStep          "auto"            % 最小计算步长
MaxNumMinSteps       "-1"          % 最小步长最多允许次数，-1表示无限制
InitialStep          "auto"        % 初始步长选择方式
FixedStep        "auto"            % 固定步长
MaxOrder         5                 % 最大阶次
OutputOption         "RefineOutputTimes"   % 以下为仿真结果返回变量设置
OutputTimes          "[]"          % 输出时间
LoadExternalInput    off           % 不调用外部输入信号
ExternalInput        "[t, u]"      % 外部信号变量名
SaveTime         on                % 返回时间变量
TimeSaveName         "tout"        % 设置时间向量的变量名
SaveState        off               % 不返回系统的状态变量
StateSaveName        "xout"        % 状态变量的变量名
SaveOutput           on            % 返回系统的输出变量
OutputSaveName       "yout"        % 设置输出变量名
LoadInitialState     off           % 不读入初始状态
InitialState         "xInitial"    % 初始状态变量名
SaveFinalState       off           % 不存最终状态变量的值
```

```
FinalStateName        "xFinal"      % 最终状态变量名
SaveFormat            "Array"       % 返回变量的数据格式
LimitDataPoints       on            % 限制输出的最多点数
MaxDataPoints         "1000"        % 最多输出点数
Decimation            "1"           % 小数模式设置
AlgebraicLoopMsg      "warning"     % 警告信息设置：代数环警告
MinStepSizeMsg        "warning"     % 小于最小步长时警告
UnconnectedInputMsg       "warning" % 含有未连接输入口的警告
UnconnectedOutputMsg      "warning" % 含有未连接输出口的警告
UnconnectedLineMsg        "warning" % 含有未连接连线的警告
InheritedTsInSrcMsg       "warning" % 继承源信号采样周期，
... ...                             % 以下警告信息选项从略
Profile          off                %
SimulationMode        "normal"      % 仿真模态设置，默认为 normal
RTWSystemTargetFile       "grt.tlc"   % 以下为 Simulink 实时工具的内容，从略
... ...
AccelSystemTargetFile    "accel.tlc"  % 以下为加速运行模式的选项
AccelTemplateMakefile    "accel_default_tmf"
AccelMakeCommand      "make_rtw"
TryForcingSFcnDF      off
ExtModeMexFile        "ext_comm"      % 以下为外部运行模式的选项，从略
... ...
BlockDefaults {                       % 默认的模块设置
  Orientation           "right"       % 模块方向
  ForegroundColor       "black"       % 模块前景颜色
  BackgroundColor       "white"       % 模块背景颜色
  DropShadow            off           % 模块是否带阴影
  NamePlacement         "normal"      % 模块名称的位置
  FontName              "Helvetica"   % 模块字体
  FontSize              10            % 模块字号大小
  FontWeight            "normal"      % 字体是否加重
  FontAngle             "normal"      % 字体是否倾斜
  ShowName              on            % 是否显示模块名称
}
AnnotationDefaults {                  % 默认的模块标注方式
  HorizontalAlignment       "center"  % 模块标记的水平对齐方式
  VerticalAlignment         "middle"  % 模块标记的垂直对齐方式
  ForegroundColor       "black"       % 模块图标的前景颜色
```

```
      BackgroundColor     "white"     % 模块图标的背景颜色
      DropShadow          off         % 模块是否要阴影
      FontName            "Helvetica" % 模块图标字体
      FontSize            10          % 模块图标字号大小
      FontWeight          "normal"    % 字体是否加重
      FontAngle           "normal"    % 字体是否倾斜
    }
    LineDefaults {                    % 默认的连线设置
      FontName            "Helvetica" % 字体设置
      FontSize            9           % 字号设置
      FontWeight          "normal"    % 是否加重
      FontAngle           "normal"    % 是否倾斜
    }
    System {                          % 默认的系统参数设置
      Name            "newmodel"                % 模型的名称
      Location            [480, 85, 980, 386]   % 窗口位置
      Open            on                        % 打开状态
      ModelBrowserVisibility  off         % 是否需要打开模型浏览器
      ModelBrowserWidth       200         % 模型浏览器的宽度
      ScreenColor         "automatic"     % 模型窗口的颜色
      PaperOrientation        "landscape" % 模型打印方向和下面打印信息设置
      PaperPositionMode       "auto"      % 送纸模式
      PaperType           "A4"            % 纸张大小
      PaperUnits          "centimeters"   % 打印单位
      ZoomFactor          "100"           % 放大系数
      ReportName          "simulink-default.rpt" % Simulink 报告文件名
    }
  }
```

可以看出，该文件含有 Simulink 模型中所有的属性，这些属性大多数是没有必要人为修改的。如果需要修改参数，则最好通过对话框的形式来修改，如果实在想手动修改其中的某些属性，则可以使用 set_param() 函数来完成，该函数的详细调用格式将在后面给出。

函数 close_system() 允许关闭一个打开的 Simulink 模型，该函数在调用时应该给出要关闭的函数名，如果想关闭的窗口被改动且未存盘，则将给出警告对话框，通知用户将修改的模型存盘，如何才能关闭该模块。如果给出

```
close_system('MyModel',1)
```

则将强制退出该模块，并放弃新编辑的内容。

6.1.3 用语句绘制方框图

从前面几章的叙述中可以看出，以框图表示的系统可以容易地由 Simulink 提供的绘图方法绘制出来，绘制过程也是很直观方便的，但在某些特定的应用中，用户可能更希望采用 MATLAB 语句的方式直接绘制出 Simulink 的框图。

和直接绘制方法相似，用 MATLAB 语句绘制 Simulink 框图需要以下几个步骤：

(1) **建立新的模型**

前面介绍过，新模型可以由new_system() 和 open_system() 两个函数配合使用而建立起来。

```
>> new_system('newmodel'); open_system('newmodel');
```

建立了 Simulink 模型后，可以用 set_param() 函数来设置有关的参数，该函数的调用格式为：

```
set_param(f, 属性名1, 属性值1, ···)
```

其中，f 为 Simulink 模型名，“属性名”是该模型所支持属性的名称，而“属性值”是该属性允许的取值。常用的属性名可以由 simget() 函数显示出来：

```
>> f1=simget('newmodel')
f1 =
            AbsTol: 'auto'       % 绝对误差容限
             Debug: 'off'        % 是否允许跟踪调试
        Decimation: 1  % 输出位数，如果选择 n，则每隔 n 点输出 1 次
      DstWorkspace: 'current'    % 输出量工作空间
    FinalStateName: ''           % 状态变量名字
         FixedStep: 'auto'       % 定步长
      InitialState: []           % 初始状态向量
       InitialStep: 'auto'       % 初始步长
          MaxOrder: 5            % 最高算法阶次
        SaveFormat: 'Array'      % 变量类型
     MaxDataPoints: 1000         % 最大返回点数
           MaxStep: 'auto'       % 最大步长
           MinStep: []           % 最小步长
      OutputPoints: 'all'        % 输出点
   OutputVariables: 'ty'         % 输出变量
            Refine: 1            % 插值点
            RelTol: 0.0010       % 相对误差
            Solver: 'ode45'      % 选择算法
      SrcWorkspace: 'base'       % 输入量工作空间
             Trace: ''           % 是否逐步显示
```

```
        ZeroCross: 'on'           % 检测过零点
```

假如想修改该仿真模型的最小和最大步长 (即 MinStep 和 MaxStep)，则可以使用下面的语句：

```
>> f1=gcs;  % 获得当前模型的句柄
   set_param(f1,'MinStep','1e-5','MaxStep','1e-3');
```

注意，这里最小步长和最大步长不能采用数值的格式，而必须用字符的格式给出。这样设置后，将影响该模型整体的设置，例如该模型参数设置对话框将反映出来这样的设置，如图 6-1 所示。

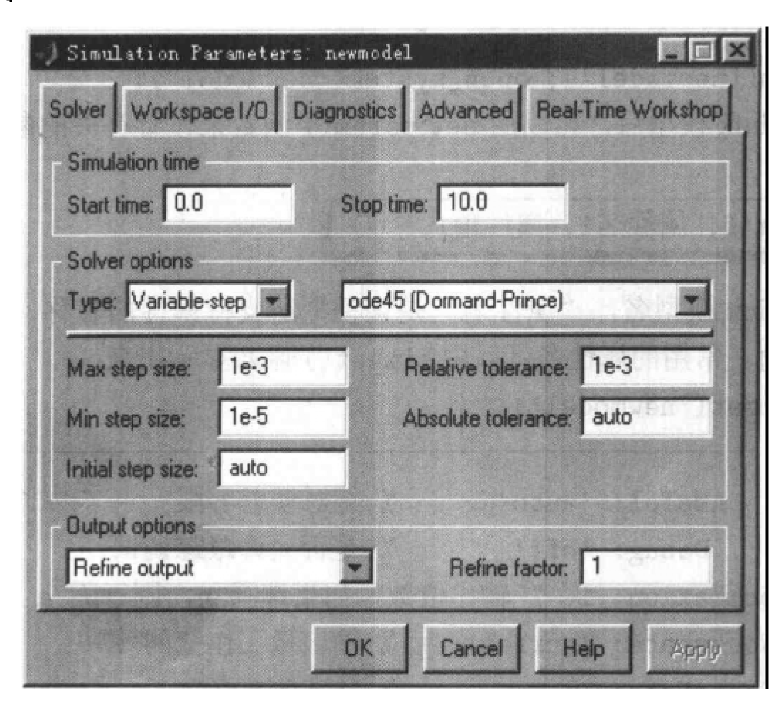


图 6-1　模块的参数设置对话框

〖例 6.2〗 *考虑第 5.3.5 节中给出的 F-14 战斗机仿真例子，在该例中，每次启动仿真之前应该运行 c5f14dat.m 程序来将模型数据读入 MATLAB 的工作空间，这样的过程还是很麻烦的，如果忘记先运行该文件，则该 Simulink 模型将给出错误信息。这里利用整个模型的 PreLoadFcn (调用前读入数据函数) 属性将有关预执行文件名关联起来*

```
>> set_param('c5f14','PreLoadFcn','c5f14dat');
   save_system('c5f14')  % 将变换后的框图存盘
```

执行过上面的语句后，应该将更新后的模型存盘，这样以后再启动该模型时，就没有必要手动执行 c5f14dat.m 程序了，因为该程序将在启动时被自动调用，将所有数据先读入 MATLAB 的工作空间。同样的设置也可以用界面完成，选中该模型的 File | Model parameters 菜单项，则可以得出如图 6-2 所示的对话框，在对话框的 Callback 标签中的 PreLoadFcn 栏目内填写预先需要调用的函数即可。使用该对话框还可以设置其他回调函数。

(2) **添加模块**

可以使用 add_block() 函数来在一个打开的模型窗口中建立新的模块，该函数的调

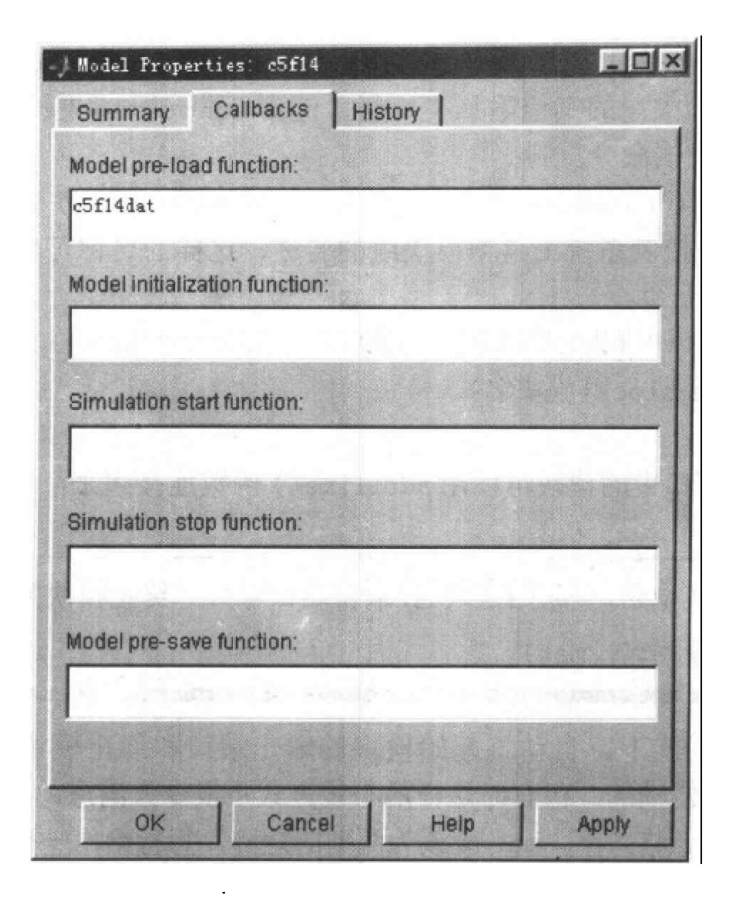


图 6-2 模型属性设置对话框

用格式为：

```
add_block(源模块名, 目标模块名, 属性名1, 属性值1, ···)
```

其中，“源模块名”为一个已知的库模块名，或在其他模块窗口中定义的模块名，例如 `'built-in/Clock'` 表示内在模块中的 `Clock` 模块，Simulink 自带的不是通过封装形成的模块都应该认为是内在模块。目标模块名则表明在目标模型窗口中使用的模块名。例如，下面的命令可以在模块窗口 newmodel 中添加一个名为 My Clock 的模块，

```
>> add_block('built-in/Clock','newmodel/My Clock');
```

在添加模块时常用的属性为：

- `'Position'` 属性表示模块的位置，应该填写 $[x_{\min}, y_{\min}, x_{\max}, y_{\max}]$，表示该模块的左下角坐标和右上角坐标，Simulink 会自动检测给出的模块位置的数值是否有效，例如它要求 $x_{\min} < x_{\max}$ 等，否则给出错误信息。
- `'Name'` 属性为该模块的名称，可以为任意字符串，还可以用 `\n` 表示换行。
- 每个模块应该有自己的参数属性，例如从图 4-20 (a) 给出的对话框中，传递函数参数的属性名为 Numerator 和 Denominator，如果想给这个模块添加数据，则应该将这两个属性设置相应的向量。

另外，还可以用模块组的方式来表示模块，例如 `'simulink3/Sources/Clock'` 可以

表示 Simulink 的 Sources 组中的 Clock 模块。除了这些模块，在 Simulink 中的 Blocksets & Toolboxes 组中还有大量的其他附加模块组，例如 Controls Toolbox 组中的线性时不变 (LTI) 模块可以用下面的命令复制：

```
>> add_block('cstblocks/LTI System','newmodel/tf')
```

其中的 cstblocks 为控制系统工具箱模块组的名称，这样的名称可以双击该模块组，从模型库窗口的标题栏获得。

如果想从模型窗口中删除某个模块，则可以用 delete_block() 函数来完成，该函数的定义格式为 delete_block(模块名)。

(3) **连接模块**

Simulink 模型窗口中的模块可以用 add_line() 函数连接起来，该函数可以有下面两种调用格式：

```
add_line(模型名，'起始模块名/输出端口号'，'终止模块名/输入端口号')
add_line(模型名，mat)
```

在第 1 种调用格式下，自动将起始模块的输出端口和终止模块的输入端口连接起来，布线的方式也是自动的。但有时这样的布线方式不很令人满意，所以可以采用第 2 种格式来进行连线，在该格式下，mat 应该为两列的矩阵，每行给出一个转折点的坐标。

如果想删除某根连线，则可以用 delete_line() 函数来完成，该函数的调用格式与 add_line() 完全一致。

(4) **模块参数修改**

模块参数可以使用 set_param() 函数来完成，该函数的调用格式为：

```
set_param(模块名，属性名1，属性值1，···)
```

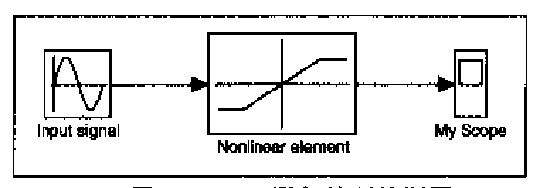
其中属性名和属性值的说明如前面所述。该函数既可以修改模块窗口的参数，也可以修改某个具体模块的参数。将在后面陆续用例子来演示该函数的使用方法。还可以使用 get_param() 函数来读取模型的某个参数。

【例 6.3】 假设想在一个新模型窗口中建立一个模型，其中有三个模块：一个正弦输入模块，其输出信号连接到一个饱和非线性环节上，饱和非线性环节的输出信号连接到一个示波器上。可以采用下面的命令来完成这样的工作：

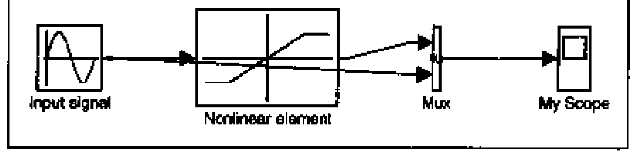
```
>> new_system('c6msys1'); open_system('c6msys1');
   set_param('c6msys1','Location',[100,100,500,400]);
   set_param('c6msys1','PaperOrientation','portrait');
   add_block('built-in/Sine Wave','c6msys1/Input signal');
   add_block('built-in/Saturation','c6msys1/Nonlinear element');
   add_block('built-in/Scope','c6msys1/My Scope');
   set_param('c6msys1/Input signal','Position',[40, 80, 80, 120]);
   set_param('c6msys1/Nonlinear element','Position',[140, 70, 230, 130]);
```

```
set_param('c6msys1/My Scope','Position',[290, 80, 310, 120]);
add_line('c6msys1','Input signal/1','Nonlinear element/1');
add_line('c6msys1','Nonlinear element/1','My Scope/1');
```

这样得出的框图如图 6-3 (a) 所示。应该说，这样的模型绘制语句还是很好理解的，首先建立了新模型窗口，设置了其位置和走纸方向属性，然后从 Simulink 内在库中选择了三个所需元件，并将其绘制到模型窗口中，并赋以新的模块名，最后将这些模块连接起来。



(a) 用 Simulink 语句绘制的框图



(b) 修改的框图

图 6-3 用 Simulink 语句绘制框图

还可以用语句修改原始模型，例如若想同时用示波器显示输入和输出信号，则可以先删除非线性环节到示波器的连线，再在其中间添加一个 Mux 模块，将非线性环节的输出和正弦输入模块的输出端口分别连接到它的两路输入口上，最后将其输出口连接到示波器上即可。可以给出如下的 Simulink 命令来完成这样的变化，这时得出的框图如图 6-3 (b) 所示。

```
>> delete_line('c6msys1','Nonlinear element/1','My Scope/1'); % 先删去连线
   set_param('c6msys1/My Scope','Position',[370,80,390,120]); % 右移示波器
   add_block('built-in/Mux','c6msys1/Mux',...                 % 添加 Mux
      'Position',[290,80,295,120],'Inputs','2');
   add_line('c6msys1','Nonlinear element/1','Mux/1');        % 自动布三条线
   add_line('c6msys1','Input signal/1','Mux/2');
   add_line('c6msys1','Mux/1','My Scope/1');
```

从得出的框图 (图 6-3 (b)) 可以看出，这样自动的布线方式有时是不理想的，在这样的情况下，需要指定转折点的坐标，例如若给出如下的命令

```
>> delete_line('c6msys1','Input signal/1','Mux/2');   % 先删去连线
   add_line('c6msys1',[100,100; 100,150; 250,150; 250,110; 290,110]);
```

则将得出如图 6-4 所示的框图，可以看出用这样的命令可以任意地连接两个模块。但首先得确定出各个转折点的坐标，这有时实现起来还是较烦琐的。

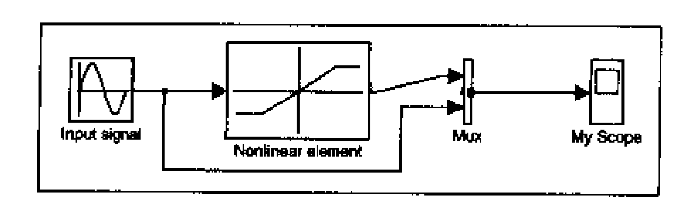


图 6-4 修改连线后的 Simulink 框图

现在可以用下面的语句移动 Mux 模块

```
>> set_param('c6msys1/Mux','position',[310,110,315,140]);   % 移动 Mux 模块
```

模块移动后的系统 Simulink 模型如图 6-5 所示，可以看出，这样移动后，原来连好的线将跟着发生变化，尽管从正弦信号到 Mux 的连线是手工完成的，一旦连接成功，Simulink 将自动确立它和两个模块之间的关系，故在其中一个位置发生变化时仍能保证正确的连接关系。

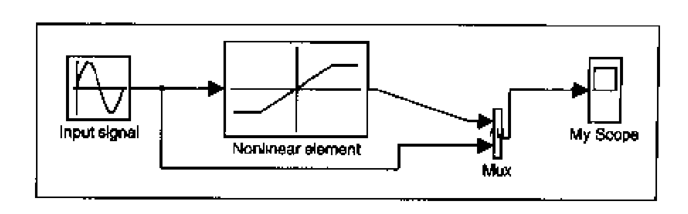


图 6-5　模块移动后的 Simulink 框图

现在可以对非线性模块的参数进行设置，假设想将其饱和下限为 -0.3，上限为 0.8，则必须得知道这些属性名，双击饱和非线性模块，可以发现这两个参数的属性名分别为 Lower limit 和 Upper limit，所以可以使用下面命令来进行设置

```
>> set_param('c6msys1/Nonlinear element',...
       'Lower limit','-0.3','Upper limit','0.8')
```

注意，给参数设置属性值时应该使用字符串，如使用 '-0.3'，而不能直接使用 -0.3。

6.2　系统仿真与线性化

6.2.1　仿真过程的命令化

启动仿真过程除了使用相应的 Simulation 菜单外，还可以使用 sim() 函数来完成，该函数的调用格式为：

```
[t, x, y] = sim(f1, tspan, options, ut)
```

其中 f1 为 Simulink 的模型名，tspan 为仿真时间控制变量，它可以为 [t0,tf]，亦即为仿真的起始和终止时间。如果 tspan 为标量，则表示终止仿真时间。参数 options 为模型控制参数，ut 为外部输入向量。该函数返回的 t 为时间列向量，x 为状态变量构成的矩阵，其中第 i 列为第 i 状态变量的时间响应，y 为输出信号构成的矩阵，也是每列对应一路输出信号。从这样的调用格式下可见，该函数是很类似于 MATLAB 中的 ode45() 等函数的。

〖例 6.4〗 假设已经建立了 F-14 系统仿真模型 c5f14.mdl，可以使用下面的命令来启动仿真过程

```
>> c5f14dat; [t,x,y]=sim('c5f14',[0,10]);
```

其中，该语句中设定 tspan 即可，无需设置其他参数。事实上，在前面的过程中已经将 c5f14dat.m 文件赋给了 c5f14.m 文件的自调用功能，所以该文件的调用也不是必需的。

〖例 6.5〗 考虑例 4.1 中给出的 Van der Pol 方程的 Simulink 模型，我们知道，当 $\mu = 1000$ 时，系统为坏条件问题，这样就应该在求解问题时选择相应的算法。选择 ode15s 算法能较好地解决问题。为了方便地对系统进行仿真，可以重新绘制出如图 6-6 所示的框图，并用下面的语句进行参

数的初始化

```
>> mu=1000; x01=1; x02=2;
```

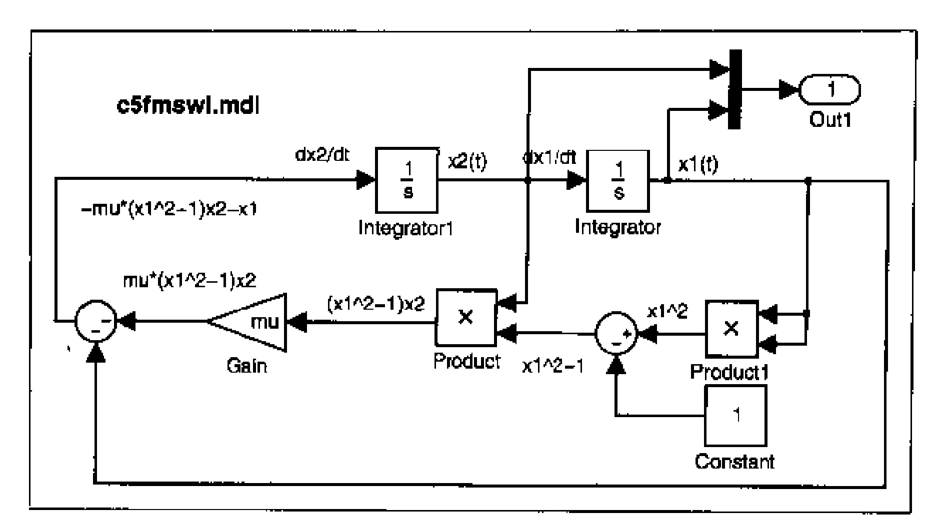


图 6-6 修改输出形式后 Simulink 框图

可以用 simget() 函数获得仿真的默认控制参数，再修改其中的 Solver 属性，则可以用下面的语句对该系统进行仿真，并绘制出系统的仿真曲线，得出的结果和图 3-6 中的完全一致。

```
>> f=simget('c6mvdp');  % 首先获得默认的控制参数
   f.Solver='ode15s';   % 选择仿真算法为 ode15s
   [t,x,y]=sim('c6mvdp',[0,3000],f); % 在新选项下进行仿真
   plot(t,y(:,1)), figure; plot(t,y(:,2))
```

所以说，用界面可以获得的结果用语句同样可以得到，且语句的格式更适合于嵌入其他程序。另外，用命令行格式进行系统仿真后，MATLAB 工作空间中将不再出现 tout 和 yout 两个变量，参数的返回应该在函数调用语句直接获得。

6.2.2 非线性模型的线性化

比较起非线性系统来说，线性系统更易于分析与设计，然而在实际应用中经常存在非线性系统，严格说来，所有的系统都含有不同程度的非线性成分。在这样的情况下，经常需要对非线性系统进行某种线性近似，从而简化系统的分析与设计过程。系统的线性化 (linearization) 是提取线性系统特征的一种有效方法。系统的线性化实际上是在系统的工作点附近的邻域内提取系统的线性特征，从而对系统进行分析设计的一种方法。

考虑下面给出的非线性系统的一般格式

$$\dot{x}_i(t) = f_i(x_1, x_2, \cdots, x_n, \boldsymbol{u}, t),\ \ i = 1, 2, \cdots, n \tag{6.1}$$

所谓系统的工作点，就是当系统状态变量导数趋于 0 时的状态变量的值。系统的工作点可以通过求取式 (6.1) 中非线性方程的方法得出

$$f_i(x_1, x_2, \cdots, x_n, \boldsymbol{u}, t) = 0,\ \ i = 1, 2, \cdots, n \tag{6.2}$$

该方程可以采用数值算法求解。在 $\boldsymbol{u}_0$ 输入信号作用下，得到工作点 $\boldsymbol{x}_0$ 后，非线性系统在此工作点附近可以近似地表示成

$$\Delta \dot{x}_i = \sum_{j=1}^{n} \left.\frac{\partial f_i(\boldsymbol{x},\boldsymbol{u})}{\partial x_j}\right|_{\boldsymbol{x}_0,\boldsymbol{u}_0} \Delta x_j + \sum_{j=1}^{p} \left.\frac{\partial f_i(\boldsymbol{x},\boldsymbol{u})}{\partial u_j}\right|_{\boldsymbol{x}_0,\boldsymbol{u}_0} \Delta u_j \tag{6.3}$$

可以将系统模型写成

$$\Delta\dot{\boldsymbol{x}}(t) = \boldsymbol{A}_l \Delta \boldsymbol{x}(t) + \boldsymbol{B}_l \Delta \boldsymbol{u}(t) \tag{6.4}$$

其中 Jacob 矩阵

$$\boldsymbol{A}_l = \begin{bmatrix} \partial f_1/\partial x_1 & \cdots & \partial f_1/\partial x_n \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1 & \cdots & \partial f_n/\partial x_n \end{bmatrix}, \quad \boldsymbol{B}_l = \begin{bmatrix} \partial f_1/\partial u_1 & \cdots & \partial f_1/\partial u_p \\ \vdots & \ddots & \vdots \\ \partial f_n/\partial u_1 & \cdots & \partial f_n/\partial u_p \end{bmatrix} \tag{6.5}$$

令新的状态变量和输入变量分别为 $\boldsymbol{z}(t) = \Delta\boldsymbol{x}(t)$ 和 $\boldsymbol{v}(t) = \Delta\boldsymbol{u}(t)$，则可以将系统的状态方程写作

$$\dot{\boldsymbol{z}}(t) = \boldsymbol{A}_l \boldsymbol{z}(t) + \boldsymbol{B}_l \boldsymbol{v}(t) \tag{6.6}$$

则该模型称为原系统的线性化模型。

Simulink 中提供了 `trim()` 函数，可以用来求解系统在指定输入下的工作点，该函数的调用格式为：

```
[x,u,y,dx]=trim(model_name,x0,u0)
```

其中 model_name 为 Simulink 模型的文件名，变量 x0，u0 为数值算法所要求的起始搜索点，是用户应该指定的状态初值和工作点的输入信号。对不含有非线性环节的系统来说，则不需要初始值 x0, u0 的设定。调用函数之后，实际的工作点在 x, u, y 变量中返回，而状态变量的导数值在变量 dx 中返回。从理论上讲，状态变量在工作点处的一阶导数都应该等于 0，该函数正是基于这样的假设，采用数值最优化算法而实现的。

〖例 6.6〗 考虑如图 6-7 所示的非线性系统模型。可以看出，该系统中有两个非线性环节，用 Simulink 程序就可以容易地画出该系统的仿真框图模型，如图 6-8 所示。注意，所建的模型中分别用输入和输出端子来表示原系统的输入和输出。

可以用下面的命令获得系统默认状态下的工作点。

```
>> [x,u,y,dx]=trim('c6nlsys')
x =

     0
     0
     0

u =

     0
```

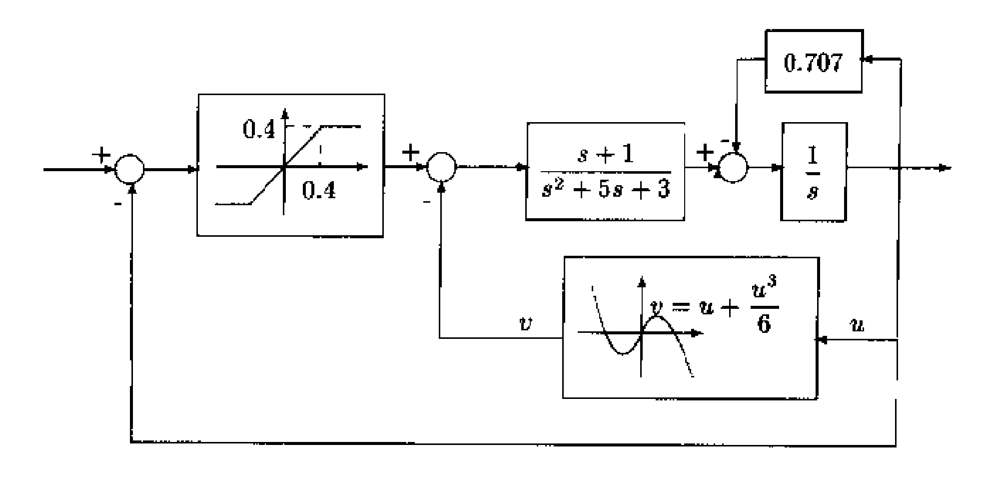


图 6-7 非线性系统模型框图

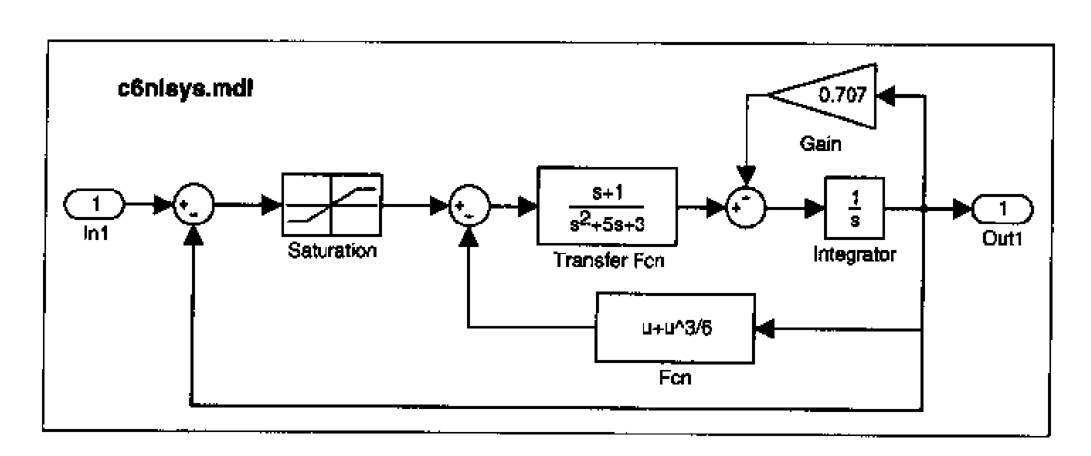


图 6-8 非线性系统模型的 Simulink 表示

```
y =
     0
dx =
     0
     0
     0
```

可以看出，这样当然能获得系统的工作点，但工作点的输入信号为 0，一般情况下更希望获得阶跃输入下的工作点，可以给出下面的命令:

```
>> [x,u,y,dx]=trim('c6nlsys',[],1)
x =
    0.1281
    0.0000
    0.0905
u =
     1
```

```
y =
    0.1281
dx =
   1.0e-009 *
   -0.0001
   -0.8055
    0.0000
```

这样就能获得系统在阶跃输入下的工作点。

获得了工作点后，可以采用 Simulink 程序中提供的 `linmod2()` 函数来求取系统的线性化模型，该函数的调用格式为：

```
[A,B,C,D]=linmod2(model_name,x,u)
```

其中 x, u 为工作点处的状态变量与输入，得出的线性化的状态方程模型 在 (A, B, C, D) 变量中返回。若省略了 x, u 变量 (工作点变量)，则将得出默认的线性化模型。事实上，若 Simulink 模型完全由线性环节搭建，则可以使用此函数提取出整个系统的线性模型，这时也无需输入工作点变量。

〖例 6.7〗 考虑例 4.2 中给出的直流电机拖动系统，前面已经建立起了 Simulink 模型来描述该系统，相应的 Simulink 文件为 c4mex2.mdl，可以用下面的语句获得该模型的零极点形式

```
>> [a,b,c,d]=linmod2('c4mex2'); % 提取系统线性模型的状态方程参数
   G=zpk(ss(a,b,c,d));  % 求取系统的因式分解后传递函数
   minreal(G)           % 求出最小实现模型 (消去相同的零极点)
Zero/pole/gain:
                      1118525021.9491 (s+5.882) (s+6.667)
------------------------------------------------------------------------
(s+179.6) (s+98.9) (s+8.307) (s^2 + 0.8936s + 5.819) (s^2 + 68.26s + 2248)
```

〖例 6.8〗 考虑第 5.3.5 节中给出的 F-14 战斗机的例子，可以看出，因为该系统中只含有线性环节，故可以 `linmod2()` 函数来提取其相应的线性系统模型，例如可以给出如下语句：

```
>> [a,b,c,d]=linmod2('c5f14'), G=ss(a,b,c,d);
```

则将立即得出系统的线性状态方程模型参数如下

```
a =
  1.0e+003 *
 -0.0006  0.6894  0.0021  0.0005       0 -1.2800       0       0       0       0
 -0.0000 -0.0007  0.0000  0.0000 -0.0001 -0.1377       0       0       0       0
       0       0 -0.0008 -0.0002       0       0       0       0       0       0
       0       0  0.0010       0       0       0       0       0       0       0
       0       0  0.0033  0.0007 -0.0085       0       0       0       0       0
       0  0.0014       0       0       0 -0.0200 -0.0017  0.0030 -0.0175  0.0010
       0  0.0010       0       0       0       0 -0.0041       0       0       0
```

```
  0.0000       0       0       0       0       0       0 -0.0025       0       0
       0       0       0       0       0       0       0       0 -0.0100       0
       0  0.0014       0       0       0       0 -0.0017  0.0030 -0.0175       0
b =
       0       0
       0       0
       1       0
       0       0
       0       0
       0       0
       0       0
       0       0
       0       1
       0       0
c =
       0 21.4099       0       0       0       0       0       0       0       0
  0.0015       0       0       0       0       0       0       0       0       0
       0  1.0000       0       0       0       0       0       0       0       0
d =
       0       0
       0       0
       0       0
```

〖例 6.9〗 考虑例 6.6 中给出的非线性系统模型，首先得出系统的工作点，然后在工作点附近对系统的线性化模型：

```
>> [x,u,y,dx]=trim('c6nlsys'); % 零输入工作点计算
   [a,b,c,d]=linmod2('c6nlsys',x,u); G=ss(a,b,c,d) % 线性化
a =
             x1       x2       x3
     x1  -0.707        1        1
     x2      -2       -5       -3
     x3       0        1        0
b =
         u1
     x1   0
     x2   1
     x3   0
c =
          x1  x2  x3
```

```
    y1   1   0   0
 d =
        u1
    y1   0
 Continuous-time model.
```

再考虑阶跃输入下的系统的工作点及其线性化模型

```
>> [x,u,y,dx]=trim('c6nlsys',[],1); %阶跃输入工作点
   [a,b,c,d]=linmod2('c6nlsys',x,u); G=ss(a,b,c,d)
 a =
              x1       x2       x3
    x1   -0.707        1        1
    x2   -1.008       -5       -3
    x3        0        1        0
 b =
        u1
    x1   0
    x2   0
    x3   0
 c =
        x1  x2  x3
    y1   1   0   0
 d =
        u1
    y1   0
 Continuous-time model.
```

可以看出，对不同的工作点可能得出不同的线性化模型。在阶跃输入的工作点处得出的状态方程模型中，$\boldsymbol{B}$ 为零向量，故得出的线性化模型可能不实用。

6.2.3 纯时间延迟系统 Padé 近似

可以利用 Padé 近似技术对时间延迟系统进行更精确的线性化，假设时间延迟环节 $\mathrm{e}^{-\tau s}$ 可以由有理传递函数的形式来近似，典型的 n 阶 Padé 近似可以表示为

$$P_{n,\tau}(s)=\frac{1-\tau s/2+p_1(\tau s)^2-p_2(\tau s)^3+\cdots+(-1)^{n+1}p_n(\tau s)^n}{1+\tau s/2+p_1(\tau s)^2+p_2(\tau s)^3+\cdots+p_n(\tau s)^n} \tag{6.7}$$

其中 $n\leq 6$ 阶的 Padé 近似的系数在表 6-1 中给出。

控制系统工具箱中给出了 pade() 函数，该函数可以求取 Padé 近似的有理传递函数

表 6-1 Padé 近似系数表

阶次 n	p_1	p_2	p_3	p_4	p_5
1	0	0	0	0	0
2	1/12	0	0	0	0
3	1/10	1/120	0	0	0
4	3/28	1/84	1/1680	0	0
5	1/9	1/72	1/1008	1/30240	0
6	5/44	1/66	1/792	1/15840	1/665280

模型，这个函数的调用格式为：

```
[nP,dP]=pade(Tau,n)
```

其中 Tau 为延迟常数 τ, n 为 Padé 阶次的阶次。Padé 有理近似 $P_{n,\tau}(s)$ 的分子和分母在 (nP, dP) 变量中返回。

〖例 6.10〗 考虑一个单位负反馈系统，其前向通路由一个线性传递函数和一个纯时间延迟构成，可以按图 6-9 的方式构造起 Simulink 框图，在框图中不但有输入和输出端子，此外从控制系统工具箱的 Simulink 库中取输入输出端子连到系统上，可以双击时间延迟模块，在其中的 Pade order (for linearization) 填写参数 3，再进行线性化，则得出如下的结果：

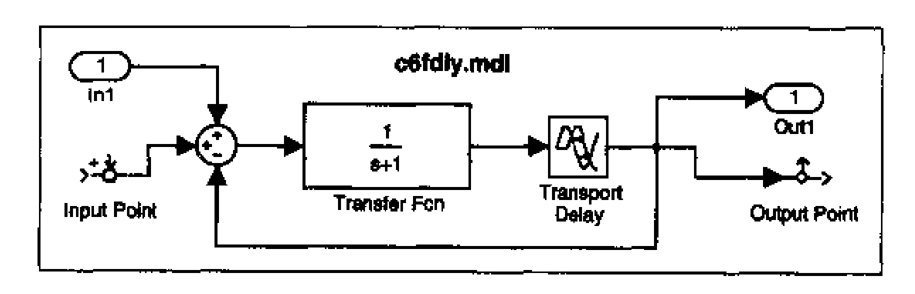


图 6-9 时间延迟系统的 Simulink 表示

```
>> [a,b,c,d]=linmod2('c6fdly')
a =
    -1
b =
     1
c =
     0
d =
     0
```

可见，linmod2() 函数并未对原始模型进行 Padé 近似，所以得出的线性化模型并不实用

(其实这样得出的传递函数为 0)，故 Simulink 中 `linmod2()` 函数在这个问题上导致错误的结论。可以选择 Tools | Linear Analysis 菜单项，这样将得出如图 6-10 所示的窗口。选择 Simulink | Get

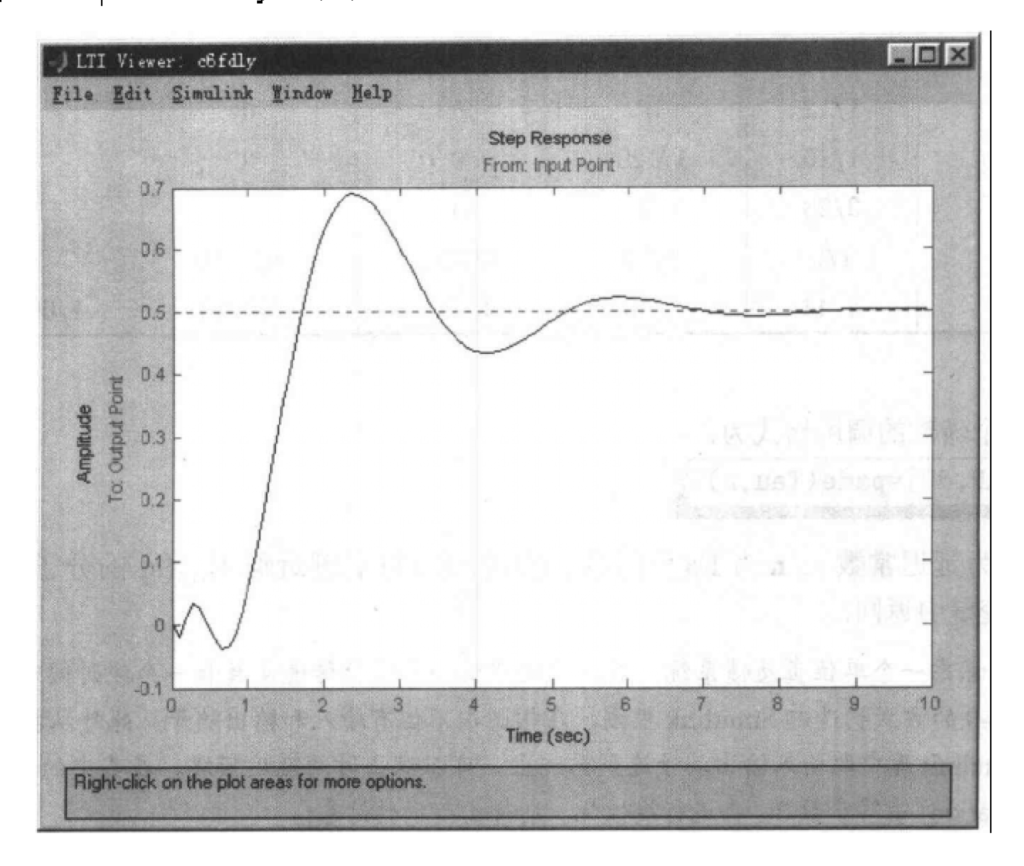


图 6-10 线性系统分析工具

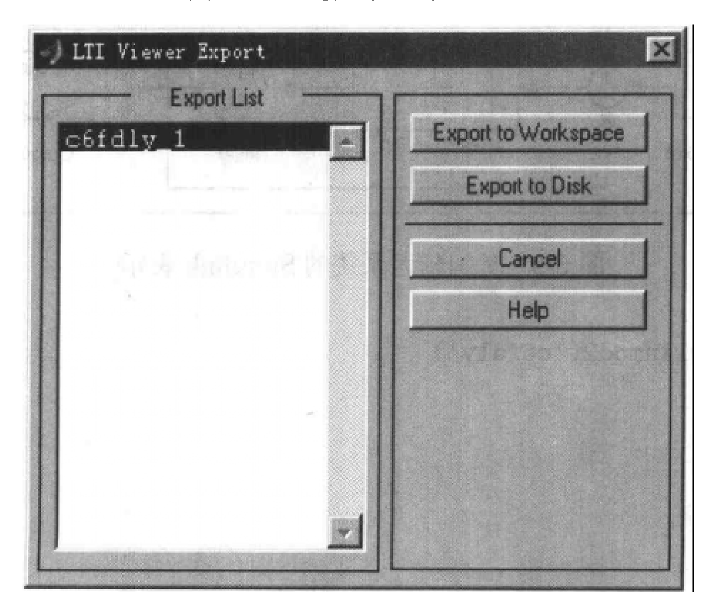


图 6-11 模型输出对话框

Linearized Model 菜单项，则将得出系统的线性化模型，并在该窗口中显示了 Padé 近似下系统的阶跃响应曲线，在此界面下还能对系统进行其他分析，如绘制系统的 Bode 图等。

如果想获得实际的线性化模型，则可以选择 File | Export 菜单项，这样将得出如图 6-11 所示的对话框。在该对话框中可以选择系统模型的输出方式，如选择 Export to workspace 则将把得出的线性化模型输出到 MATLAB 的工作空间，这样就能得出其传递函数模型为

```
>> tf(c6fdly_1)
Transfer function from input "Input Point" to output "Output Point":
                     -s^3 + 12 s^2 - 60 s + 120
---------------------------------------------------------------------
s^4 + 12 s^3 + (84-3.553e-015i) s^2 + (120-2.842e-014i) s + 240
```

由于采用了数值算法，在系统传递函数分母上出现了极其微小的虚数，略去它们即可。可以看出，这样的线性化程序明显优于 Simulink 下的 `linmod2()` 函数。

可以选择不同的 Padé 近似阶次，例如选择 $n = 2,3,4$，则可以得出不同的阶跃响应，如图 6-12 所示。可以看出，一般取 $n = 3$ 就能得出较好的近似效果。

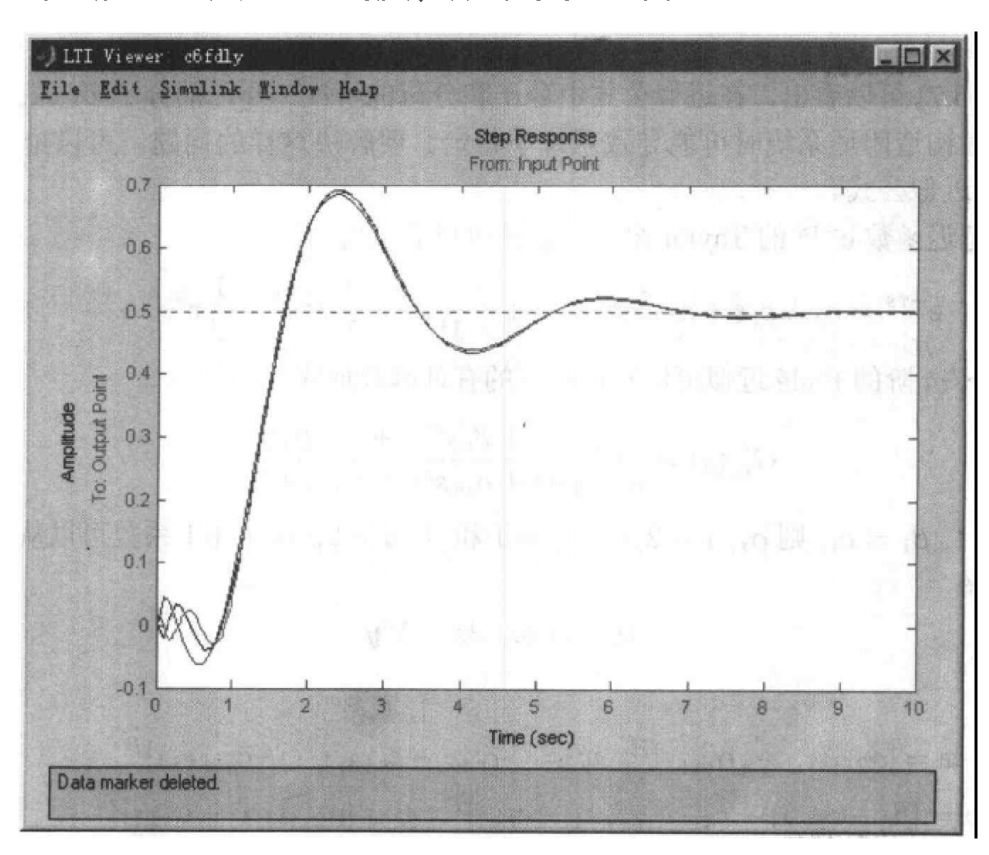


图 6-12 不同近似阶次下的阶跃响应

【例 6.11】 再考虑例 6.6 中阶跃输入下的线性化问题。用控制系统工具箱中的输入输出端口取代原来的输入输出端口，则将得出如图 6-13 所示的系统模型。对该模型采用控制系统工具箱中的线性化方法，并将得出的结果返回到 MATLAB 的工作空间，则可以得出如下的结果:

```
>> zpk(c6nlsys2_1)
Zero/pole/gain from input "Input Point" to output "Output Point":
                  (s+1)
```

```
--------------------------------
(s+3.702) (s^2 + 2.005s + 1.113)
```

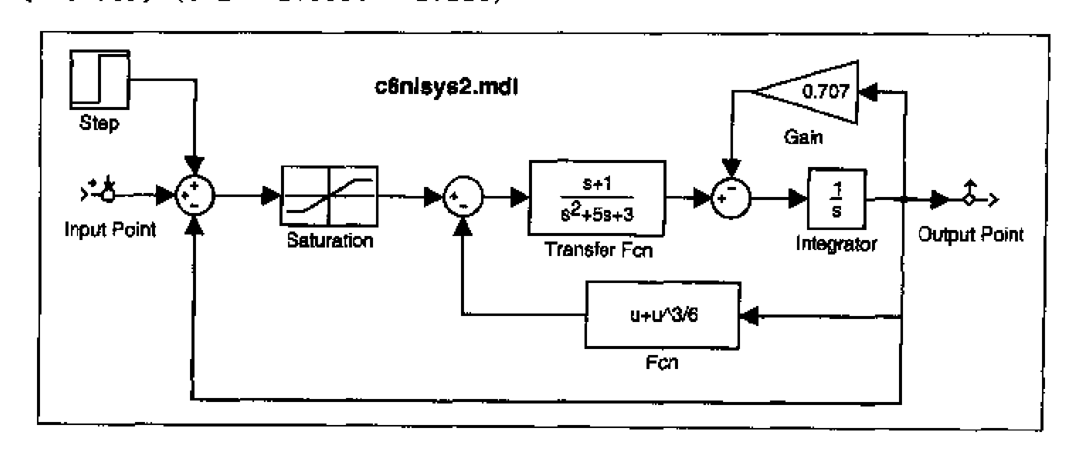


图 6-13 改进的系统的 Simulink 模型

从式 (6.7) 可以看出，在近似公式中分子和分母的阶次一样，而分子多项式含有负系数，所以在构造闭环系统时可能导致系统不稳定。要解决这样的问题，可以推导出更合适的 Padé 近似公式。

因为延迟函数 $\mathrm{e}^{-\tau s}$ 的 Taylor 幂级数展开可以表示为

$$\mathrm{e}^{-\tau s}=c_0+c_1 s+c_2 s^2+\cdots=1-\frac{1}{1!}\tau s+\frac{1}{2!}\tau^2 s^2-\frac{1}{3!}\tau^3 s^3+\cdots \tag{6.8}$$

假设 r/m 阶的 Padé 近似可以写成如下的有理函数形式

$$G_m^r(s)=\frac{\beta_{r+1}s^r+\beta_r s^{r-1}+\cdots+\beta_1}{\alpha_{m+1}s^m+\alpha_m s^{m-1}+\ldots+\alpha_1} \tag{6.9}$$

式中 $\alpha_1=1$, $\beta_1=c_1$, 则 α_i, $i=2,\cdots,m+1$ 和 β_i, $i=2,\cdots,k+1$ 系数可以从下面的方程求解出来

$$\boldsymbol{W}\boldsymbol{x}=\boldsymbol{w},\quad \boldsymbol{v}=\boldsymbol{V}\boldsymbol{y} \tag{6.10}$$

其中

$$\begin{aligned}\boldsymbol{x}&=[\alpha_2,\alpha_3,\cdots,\alpha_{m+1}]^T,\ \ \boldsymbol{w}=[-c_{r+2},-c_{r+3},\cdots,-c_{m+r+1}]^T\\ \boldsymbol{v}&=[\beta_2-c_2,\beta_3-c_3,\cdots,\beta_{r+1}-c_{r+1}]^T,\ \ \boldsymbol{y}=[\alpha_2,\alpha_3,\cdots,\alpha_{r+1}]^T\end{aligned} \tag{6.11}$$

且

$$\boldsymbol{W}=\begin{bmatrix} c_{r+1} & c_r & \cdots & 0 & \cdots & 0\\ c_{r+2} & c_{r+1} & \cdots & c_1 & \cdots & 0\\ \vdots & \vdots & \cdots & \vdots & \ddots & \vdots\\ c_{r+m} & c_{r+m-1} & \cdots & c_{m-1} & \cdots & c_{r+1}\end{bmatrix} \tag{6.12}$$

$$\boldsymbol{V}=\begin{bmatrix} c_1 & 0 & 0 & \cdots & 0\\ c_2 & c_1 & 0 & \cdots & 0\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ c_r & c_{r-1} & c_{r-2} & \cdots & c_1\end{bmatrix} \tag{6.13}$$

我们编写了一个 MATLAB 函数 paderm() 来计算时间延迟项的 Padé 有理近似，该函数的清单如下：

```
function [nP,dP]=paderm(tau,r,m)
c(1)=1;
for i=2:r+m+1, c(i)=-c(i-1)*tau/(i-1); end
w=-c(r+2:m+r+1)';
vv=[c(r+1:-1:1)'; zeros(m-1-r,1)];
W=rot90(hankel(c(m+r:-1:r+1),vv));
V=rot90(hankel(c(r:-1:1)));
x=[1 (W\w)']; y=[1 x(2:r+1)*V'+c(2:r+1)];
dP=x(m+1:-1:1)/x(m+1); nP=y(r+1:-1:1)/x(m+1);
```

该函数的调用格式为：

```
[nP,dP]=paderm(tau,r,m)
```

其中 r, m 分别为所要求的 Padé 有理近似的分子和分母阶次，这个函数调用后，Padé 近似有理式的分子和分母多项式分别由 nP 和 dP 返回。

〖例 6.12〗 考虑一个带有纯延迟的简单例子 $G(s) = e^{-s}$，由下面的 MATLAB 语句可以求出其各个阶次的 Padé 近似结果

```
>> tau=1;  % 时间延迟常数
   [nP1,dP1]=paderm(tau,1,3); GP1=tf(nP1,dP1)
Transfer function:
      -6 s + 24
-----------------------
s^3 + 6 s^2 + 18 s + 24
>> [nP2,dP2]=paderm(tau,0,3); GP2=tf(nP2,dP2)
Transfer function:
         6
---------------------
s^3 + 3 s^2 + 6 s + 6
>> [nP3,dP3]=pade(tau,3); GP3=tf(nP3,dP3)
Transfer function:
-s^3 + 12 s^2 - 60 s + 120
--------------------------
s^3 + 12 s^2 + 60 s + 120
>> step(GP1,GP2,GP3)  % 绘制出三种近似的阶跃响应曲线
   h=line([0,1,1.00001,7],[0,0,1,1]); % 画延迟模块理论值
   set(h,'LineWidth',2); % 设置其为粗线
```

并绘制出阶跃响应曲线，如图 6-14 所示。可见，用三阶近似的效果明显优于普通 Padé 近似的效

果，因为这样做能避免初始的强振荡。

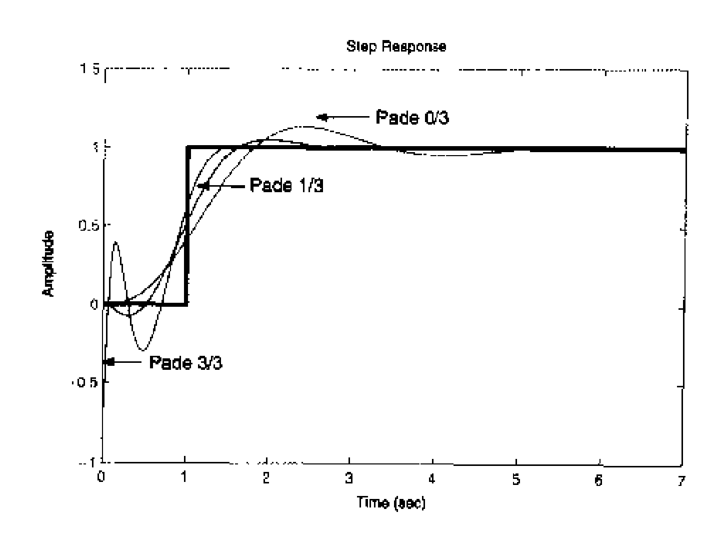


图 6-14 延迟环节的不同三阶 Padé 近似

遗憾的是，虽然这样的 Padé 近似比分子和分母阶次相同的近似有很多的优势，但在控制系统工具箱的线性化功能中没有采用这样的近似，所以近似效果有时无法改善。

6.3 S-函数的编写及应用

在实际应用中，通常会发现有些过程用普通的 Simulink 模块不容易搭建，可以使用 Simulink 支持的 S-函数格式，用 MATLAB 语言或 C 等语言写出描述过程的程序，构成 S-函数模块，像标准 Simulink 模块那样直接调用。

S-函数有固定的程序格式，用 MATLAB 语言可以编写 S-函数，此外还允许采用 C 语言、C++、Fortran 和 Ada 等语言编写，只不过用这些语言编写程序时，需要用编译器生成动态连接库 (DLL) 文件，可以在 Simulink 中直接调用。这里主要介绍用 MATLAB 语言设计 S-函数的方法，并将通过例子介绍 S-函数的应用与技巧。

6.3.1 用 MATLAB 语句编写 S-函数

有些算法较复杂的模块可以用 MATLAB 语言按照 S-函数的格式来编写，但应该注意，这样构造的 S-函数只能用于基于 Simulink 的仿真，并不能将其转换成独立于 MATLAB 的独立程序。用 C 语言格式建立的 S-函数则可以转换成独立程序。

S-函数的引导语句为：

```
function [sys,x0,str,ts]=f(t,x,u,flag,p1,p2,...)
```

其中 f 为 S-函数的函数名，t,x,u 分别为时间、状态和输入信号，flag 为标志位，其意义和有关信息在表 6-2 中给出，一般应用中很少使用 flag 为 4 和 9。在该表中还解释了在不同的 flag 下的返回参数类型。在该函数中还允许使用任意数量的附加参数 p1, p2, ⋯

表 6-2 flag 参数表

取值	功能	调用函数名	返回参数
0	初始化	mdlInitializeSizes	sys 为初始化参数，x0, str, ts 如其定义
1	连续状态计算	mdlDerivatives	sys 返回连续状态
2	离散状态计算	mdlUpdate	sys 返回离散状态
3	输出信号计算	mdlOutputs	sys 返回系统输出
4	下一步仿真时刻	mdlGetTimeOfNextVarHit	sys 下一步仿真的时间
9	终止仿真设定	mdlTerminate	无

等，这些参数可以在 S-函数的参数对话框中给出，后面将用例子演示。下面将分别介绍 S-函数的编写方法。

- **参数初始设定** 首先通过 sizes=simsizes 语句获得默认的系统参数变量 sizes。得出的 sizes 实际上是一个结构体变量，其常用成员为：
  - ◇ NumContStates 表示 S-函数描述的模块中连续状态的个数；
  - ◇ NumDiscStates 表示离散状态的个数；
  - ◇ NumInputs 和 NumOutputs 分别表示模块输入和输出的个数；
  - ◇ DirFeedthrough 为输入信号是否直接在输出端出现的标识，取值可以为 0, 1;
  - ◇ NumSampleTimes 为模块采样周期的个数，即 S-函数支持多采样周期的系统。

  按照要求设置好的结构体 sizes 应该在通过 sys=simsizes(sizes) 语句赋给 sys 参数。除了 sys 外，还应该设置系统的初始状态变量 x0、说明变量 str 和采样周期变量 ts，其中 ts 变量应该为双列的矩阵，其中每一行对应一个采样周期。对连续系统和有单个采样周期的系统来说，该变量为 [t1,t2]，其中 t1 为采样周期，如果取 t1=-1 则将继承输入信号的采样周期。参数 t2 为偏移量，一般取为 0。

- **状态的动态更新** 连续模块的状态更新由 mdlDerivatives 函数来设置，而离散状态的更新应该由 mdlUpdate 函数设置。这些函数的输出值，即相应的状态，均由 sys 变量返回。如果要仿真混杂系统 (hybrid system)，则需要写出这两个函数来分别描述连续状态和离散状态。

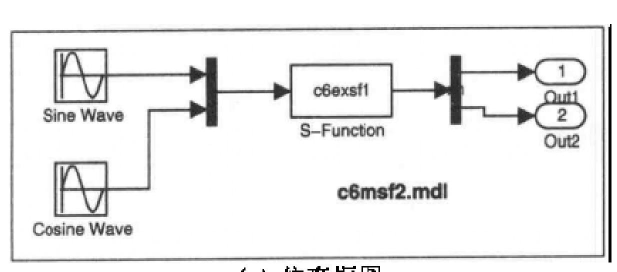
- **输出信号的计算** 调用 mdlOutputs 函数就可以计算出模块的输出信号，系统的输出仍然由 sys 变量返回。

Simulink 中提供了一个 sfuntmpl.m 的模板文件，可以从这个模板出发构建自己的 S-函数，如果需要，则将该文件复制到你的工作目录，以它为模板，就可以构建起你自己的 S-函数。其实，S-函数的结构还是很简单的，看了下面的例子就能够自己编写了。

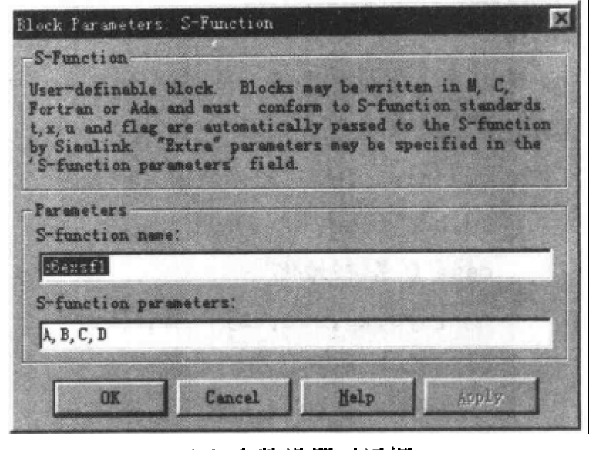
【例 6.13】 考虑例 5.2 中给出的带状态输出的线性系统模型，可以为该系统写出 S-函数的模块，其清单为

```
function [sys,x0,str,ts] = c6exsf1(t,x,u,flag,A,B,C,D) % 带附加参数 A,B,C,D
switch flag,
case 0 % 初始化设置
   [sys,x0,str,ts]=mdlInitializeSizes(A,D);
case 1 % 连续状态变量计算
   sys = mdlDerivatives(t,x,u,A,B);
case 3 % 输出量计算
   sys = mdlOutputs(t,x,u,C,D);
case { 2, 4, 9 } % 未定义标志
   sys = [];
otherwise % 处理错误
   error(['Unhandled flag = ',num2str(flag)]);
end
%==============================================================
% mdlInitializeSizes 进行初始化，设置系统变量的大小
%==============================================================
function [sys,x0,str,ts] = mdlInitializeSizes(A,D)
sizes = simsizes;  % 取系统默认设置
sizes.NumContStates = size(A,1); % 设置连续变量个数
sizes.NumDiscStates = 0; % 设置离散状态个数，因为无离散状态，故设其为 0
sizes.NumOutputs=size(A,1)+size(D,1); % 设置输出变量个数为 D 行数加系统阶次
sizes.NumInputs = size(D,2); % 设置输入变量的个数，为 D 的列数
sizes.DirFeedthrough = 1; % 输出量的计算取决于输入量 D
sizes.NumSampleTimes = 1; % 采样周期的个数，
sys = simsizes(sizes); % 设置系统的大小参数
x0 = zeros(size(A,1),1); % 设置为零初始状态
str = []; % 设置字符串矩阵
ts = [-1 0]; % 采样周期设置，前面的 -1 表示继承输入信号的采样周期
%==============================================================
% mdlDerivatives 计算系统的状态变量
%==============================================================
function sys = mdlDerivatives(t,x,u,A,B)
sys = A*x + B*u;
%==============================================================
% mdlOutputs 计算系统输出
%==============================================================
function sys = mdlOutputs(t,x,u,C,D)
sys = [C*x+D*u; x]; % 系统的增广输出
```

这样就可以建立起如图 6-15 (a) 所示的 Simulink 仿真框图。



(a) 仿真框图



(b) 参数设置对话框

图 6-15 S-函数仿真框图及参数设置

双击其中的 S-函数模块，则将得出如图 6-15 (b) 所示的参数对话框，在该对话框的 S-function name 栏目内填写 `c6exsf1`，就可以建立起该模块和我们编写的 `c6exsf1.m` 文件之间的联系，在 S-function parameters 栏目还可以给出 S-函数的附加参数 A, B, C 和 D，这些参数还可以用下面的语句在 MATLAB 命令窗口中输入

```
>> A=[2.25, -5, -1.25, -0.5;  2.25, -4.25, -1.25, -0.25;
      0.25, -0.5, -1.25,-1;  1.25, -1.75, -0.25, -0.75];
   B=[4, 6; 2, 4; 2, 2; 0, 2];
   C=[0, 0, 0, 1; 0, 2, 0, 2]; D=zeros(2,2);
```

这样再进行仿真，就能得出和例 5.2 完全一致的结果。

6.3.2 S-函数设计举例——自抗扰控制器仿真

这里主要以韩京清研究员及其合作者提出的自抗扰控制器[15] 的设计及仿真为例来演示 S-函数的编写及应用。这些内容不但能充分地演示 S-函数的使用，也能为某些系统的控制提供较好的解决方案。

〖例 6.14〗 首先考虑微分-跟踪器，其离散实现为[15]

$$\begin{cases} x_1(k+1) = x_1(k) + Tx_2(k) \\ x_2(k+1) = x_2(k) + T\mathrm{fst}(x_1(k), x_2(k), u(k), r, h) \end{cases} \tag{6.14}$$

式中，T 为采样周期，$u(k)$ 为第 k 时刻的输入信号，r 为决定跟踪快慢的参数，而 h 为输入信号被噪声污染时，决定滤波效果的参数。fst 函数可以由下面的式子计算

$$\delta = rh,\ \ \delta_0 = \delta h,\ \ y = x_1 - u + hx_2,\ \ a_0 = \sqrt{\delta^2 + 8r|y|} \tag{6.15}$$

$$a = \begin{cases} x_2 + y/h, & |y| \leq \delta_0 \\ x_2 + 0.5(a_0 - \delta)\mathrm{sign(y)}, & |y| > \delta_0 \end{cases} \tag{6.16}$$

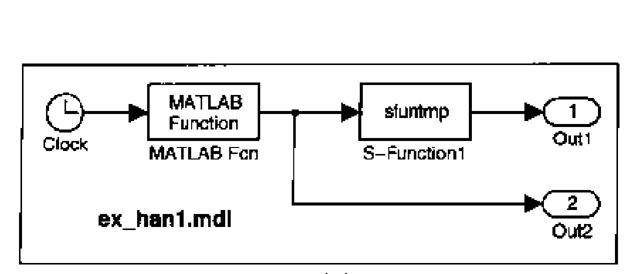
$$\text{fst} = \begin{cases} -ra/\delta, & |a| \le \delta \\ -r\text{sign}(a), & |a| > \delta \end{cases} \tag{6.17}$$

可以看出，该算法用 Simulink 模块搭建还是比较困难的，所以这里将介绍采用 S-函数建立该模块的方法。根据上述算法，立即可以写出其相应的 S-函数实现

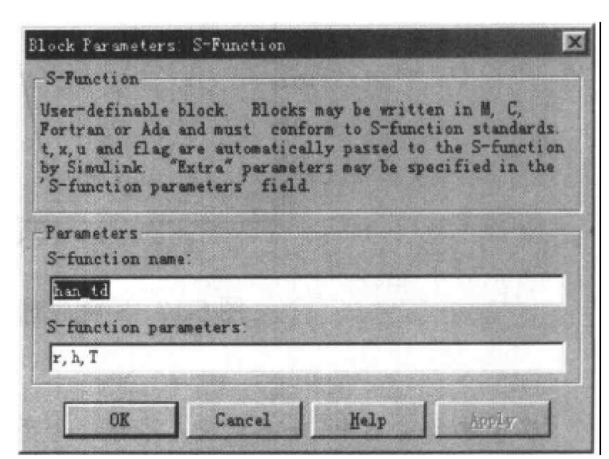
```
function [sys,x0,str,ts]=han_td(t,x,u,flag,r,h,T)
switch flag,
case 0 % 初始化
   [sys,x0,str,ts] = mdlInitializeSizes(T);
case 2 % 离散状态的更新
   sys = mdlUpdates(x,u,r,h,T);
case 3 % 输出量的计算
   sys = mdlOutputs(x);
case {1, 4, 9} % 未使用的 flag 值
   sys = [];
otherwise % 处理错误
   error(['Unhandled flag = ',num2str(flag)]);
end;
%==============================================================
% 当 flag 为 0 时进行整个系统的初始化
%==============================================================
function [sys,x0,str,ts] = mdlInitializeSizes(T)
% 首先调用 simsizes 函数得出系统规模参数 sizes，并根据离散系统的实际
% 情况设置 sizes 变量
sizes = simsizes;          % 读入初始化参数模板
sizes.NumContStates = 0; % 无连续状态
sizes.NumDiscStates = 2; % 有两个离散状态
sizes.NumOutputs = 2;     % 输出两个量：跟踪信号和微分信号
sizes.NumInputs = 1;      % 系统输入信号一路
sizes.DirFeedthrough = 0;% 输入不直接传到输出口
sizes.NumSampleTimes = 1;% 单个采样周期
sys = simsizes(sizes);    % 根据上面的设置设定系统初始化参数
x0 = [0; 0];              % 设置初始状态为零状态
str = [];                 % 将 str 变量设置为空字符串即可
ts = [-1 0];               % 采样周期，设它能继承上一级的默认值
%==============================================================
% 在主函数的 flag=2 时，更新离散系统的状态变量
%==============================================================
```

```
function sys = mdlUpdates(x,u,r,h,T)
sys(1,1)=x(1)+T*x(2);
sys(2,1)=x(2)+T*fst2(x,u,r,h);
%==============================================================
% 在主函数 flag=3 时，计算系统的输出变量：返回两个状态
%==============================================================
function sys = mdlOutputs(x)
sys=x;
%==============================================================
% 用户定义的子函数： fst2
%==============================================================
function f=fst2(x,u,r,h)
delta=r*h; delta0=delta*h; y=x(1)-u+h*x(2);
a0=sqrt(delta*delta+8*r*abs(y));
if abs(y)<=delta0
   a=x(2)+y/h;
else
   a=x(2)+0.5*(a0-delta)*sign(y);
end
if abs(a)<=delta
   f=-r*a/delta;
else
   f=-r*sign(a);
end
```

可以搭建起如图所示的仿真 6-16 (a) 框图，并存为 ex_han.mdl 文件。双击其中的 S-function



(a) 仿真框图



(b) 参数设置对话框

图 6-16 跟踪微分器的仿真框图与参数设置

模块，将给出如图 6-16 (b) 所示的 S-函数参数设置对话框，在 S-function name 栏目填写 han_td，就可以建立起该模块和我们编写的 han_td.m 文件之间的联系；在 S-function parameters 栏目还可以给出 S-函数的附加参数 r, h 和 T。在 MATLAB 工作空间中可以用 a=30, h=0.01, T=0.01 命令输入这些参数，或用

```
>> set('ex_han','PreLoadFcn','a=30; h=0.01; T=0.01;'); % 预置参数
   save_system('ex_han');                              % 存系统模型
```

命令来将有关参数直接赋给 Simulink 模型，以便每次启动该模型时都能给这些参数自动赋值。另外，可以用 MATLAB 函数的形式建立起输入信号的模块，假设我们编写了如下所示的 MATLAB 函数 han_fun()

```
function y=han_fun(x)
if x<=2*pi, % 第一个周期生成标准正弦信号
    y=sin(x);
elseif x<=2.5*pi, % 本周期内生成三角波，分为三个部分
   y=2*(x-2*pi)/pi;
elseif x<=3.5*pi, y=1-2*(x-2.5*pi)/pi;
elseif x<=4*pi, y=-2+2*(x-3*pi)/pi;
end
```

则输入信号可以选择为一个周期的正弦信号，后接一个周期的三角波信号。设置仿真的终止时间为 4π，再进行仿真，就可以用 plot(tout, yout) 命令绘制出各个信号的波形，如图 6-17 所示。可以看出，采用跟踪-微分器就可以快速地跟踪输入信号，并能立即得出该信号的微分信号。

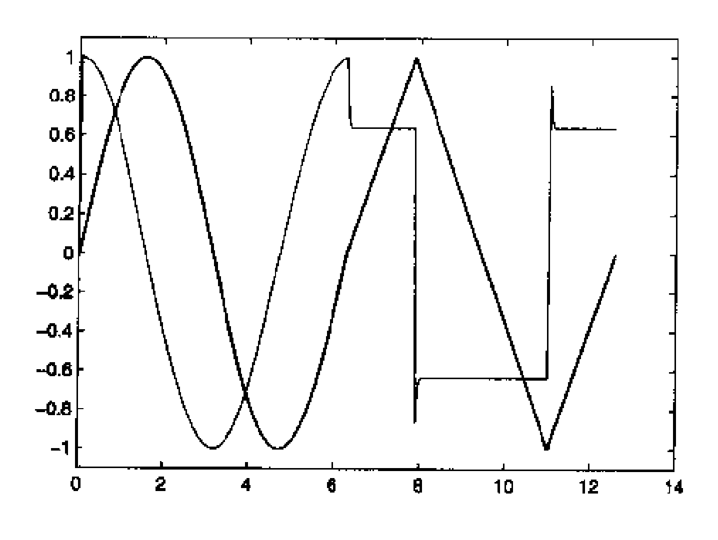


图 6-17 S-函数对给定模块的跟踪和微分

在跟踪-微分器中，r 参数决定跟踪的速度，该值越大则跟踪速度越快，同时不可避免地，在随机扰动下由较大的 r 值引起的跟踪误差也增大，这就需要增大 h 的值来抑制误差。

【例 6.15】 韩京清研究员在自抗扰控制器中首先提出了扩张的状态观测器 (extended state

observer，简称为 ESO)，其数学表示为[15]

$$\begin{cases} z_1(k+1) = z_1(k) + T[z_2(k) - \beta_{01}e(k)] \\ z_2(k+1) = z_2(k) + T[z_3(k) - \beta_{02}\text{fal}(e(k), 1/2, \delta) + bu(k)] \\ z_3(k+1) = z_3(k) - T\beta_{03}\text{fal}(e(k), 1/4, \delta) \end{cases} \tag{6.18}$$

其中 $e(k) = z_1(k) - y(k)$，且

$$\text{fal}(e, a, \delta) = \begin{cases} e\delta^{a-1}, & |e| \leq \delta \\ |e|^a \text{sign}(e), & |e| > \delta \end{cases} \tag{6.19}$$

和普通状态观测器一样，该观测器接受 $u(k)$ 和 $y(k)$ 为其输入信号。根据该数学模型，可以容易地编写出其对应的 S-函数，这里，$y(k)$ 实际上是第 2 路模块输入信号。

```
function [sys,x0,str,ts]=han_eso(t,x,u,flag,a2,d,bet,b,T)
switch flag,
case 0 % 初始化
   [sys,x0,str,ts] = mdlInitializeSizes;
case 2 % 离散状态的更新
   sys = mdlUpdates(x,u,d,bet,b,T);
case 3 % 输出量的计算
   sys = mdlOutputs(x);
case {1, 4, 9} % 未使用的 flag 值
   sys = [];
otherwise % 处理错误
   error(['Unhandled flag = ',num2str(flag)]);
end;
%=============================================================
% 当 flag 为 0 时进行整个系统的初始化
%=============================================================
function [sys,x0,str,ts] = mdlInitializeSizes
% 首先调用 simsizes 函数得出系统规模参数 sizes，并根据离散系统的实际
% 情况设置 sizes 变量
sizes = simsizes;
sizes.NumContStates = 0;  % 无连续状态变量
sizes.NumDiscStates = 3;  % 3 个离散状态变量
sizes.NumOutputs = 3;     % 三路输出
sizes.NumInputs = 2;      % 两路输入: u 和 y
sizes.DirFeedthrough = 0; % 输入信号不直接在输出中反映出来
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
```

```
x0 = [0; 0; 0];              % 设置初始状态为零状态
str = [];                    % 将 str 变量设置为空字符串
ts = [-1 0];                 % 采样周期: 假设它继承输入信号的采样周期
%==============================================================
% 在主函数的 flag=2 时，更新离散系统的状态变量
%==============================================================
function sys = mdlUpdates(x,u,d,bet,b,T)
e=x(1)-u(2);
sys(1,1)=x(1)+T*(x(2)-bet(1)*e);
sys(2,1)=x(2)+T*(x(3)-bet(2)*fal(e,0.5,d)+b*u(1));
sys(3,1)=x(3)-T*bet(3)*fal(e,0.25,d);
%==============================================================
% 在主函数 flag=3 时，计算系统的输出变量
%==============================================================
function sys = mdlOutputs(x)
sys=x;
%==============================================================
% 用户定义的子函数: fal
%==============================================================
function f=fal(e,a,d)
if abs(e)<d
   f=e*d^(a-1);
else
   f=(abs(e))^a*sign(e);
end
```

由给定的跟踪-微分器和扩张状态观测器，可以根据下面的数学工具构造起自抗扰控制器

$$\begin{cases} e_1 = v_1(k) - z_1(k), \ \ e_2 = v_2(k) - z_2(k) \\ u_0 = \beta_1 \mathrm{fal}(e_1, a_1, \delta_1) + \beta_2 \mathrm{fal}(e_2, a_2, \delta_1) \\ u(k) = u_0 - z_3(k)/b \end{cases} \tag{6.20}$$

可以看出，该控制器算法中不存在动态的过程，故可以设置连线和离散的状态个数均为 0。可以根据上面的公式编写出相应的 S-函数

```
function [sys,x0,str,ts]=han_ctrl(t,x,u,flag,aa,bet1,b,d)
switch flag,
case 0 % 初始化
   [sys,x0,str,ts] = mdlInitializeSizes(t,u,x);
case 3 % 输出量的计算
   sys = mdlOutputs(t,x,u,aa,bet1,b,d);
```

```
case {1,2,4,9} % 未使用的 flag 值
   sys = [];
otherwise % 处理错误
   error(['Unhandled flag = ',num2str(flag)]);
end;
%=============================================================
% 当 flag 为 0 时进行整个系统的初始化
%=============================================================
function [sys,x0,str,ts] = mdlInitializeSizes(t,u,x)
% 首先调用 simsizes 函数得出系统规模参数 sizes，并根据离散系统的实际
% 情况设置 sizes 变量
sizes = simsizes;
sizes.NumContStates = 0; % 连续状态数为 0
sizes.NumDiscStates = 0; % 离散状态数为 0
sizes.NumOutputs = 1;     % 输出路数为 1
sizes.NumInputs = 5;      % 输入路数为 5
sizes.DirFeedthrough = 1;% 输入在输出中直接显示出来，注意不能将其设置为 0
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = [];                  % 设置初始状态为零状态
str = [];                 % 将 str 变量设置为空字符串
ts = [-1 0];              % 采样周期： [period, offset]
%=============================================================
% 在主函数 flag=3 时，计算系统的输出变量
%=============================================================
function sys = mdlOutputs(t,x,u,aa,bet1,b,d)
e1=u(1)-u(3); e2=u(2)-u(4);
u0=bet1(1)*fal(e1,aa(1),d)+bet1(2)*fal(e2,aa(2),d);
sys=u0-u(5)/b;
%=============================================================
% 用户定义的子函数： fal
%=============================================================
function f=fal(e,a,d)
if abs(e)<d
   f=e*d^(a-1);
else
   f=(abs(e))^a*sign(e);
end
```

在该函数中，有 5 路输入信号，前两路为 z 函数 (微分-跟踪器的输出)，后三路为 v 函数 (状态观测器的输出)。系统的输出实际上就是一个控制量 $u(k)$。

假设受控对象为时变模型

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = \mathrm{sign}(\sin t) + u(t) \end{cases}$$

由这三部分构造起来的自抗扰控制器如图 6-18 所示，并保存为 ex_han2.mdl 文件，其中给

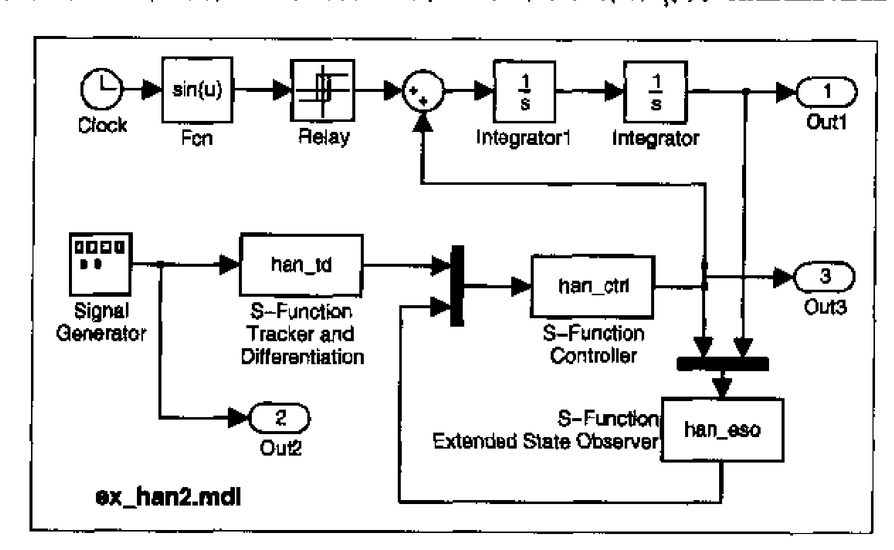


图 6-18　自抗扰控制器的仿真模型

han_eso 模块设置附加参数 d,bet,b,T, 给 han_ctrl 模块设置附加参数为 a,bet1,b,d, 并在 MATLAB 工作空间中给出如下命令:

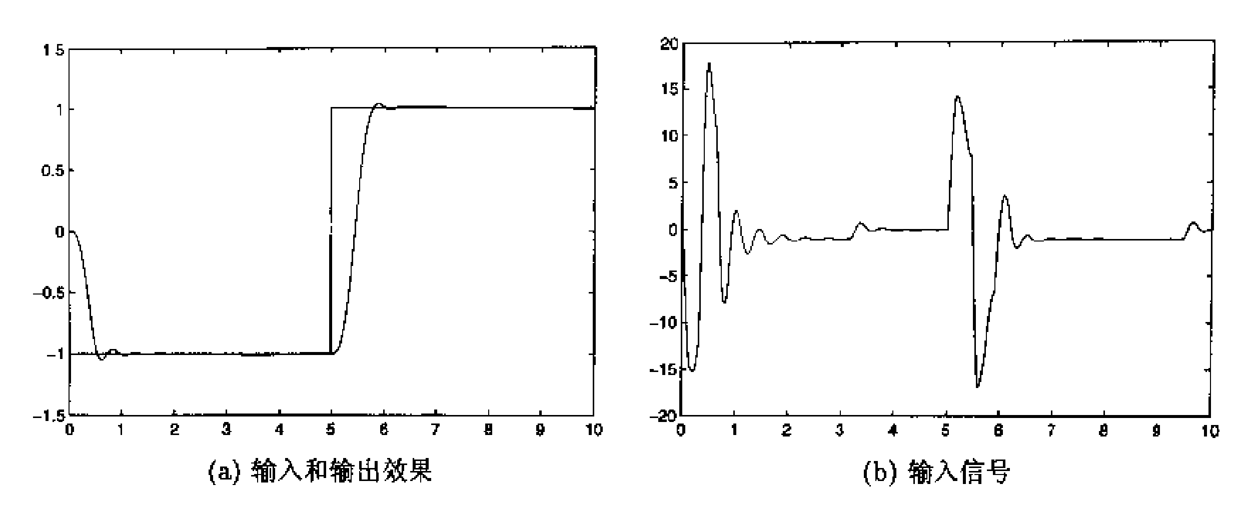
```
>> set_param('ex_han2','PreLoadFcn',['r=10; h=0.01; T=0.01; ',...
     'bet=[100,65,80]; bet1=[100,10]; aa=[0.75,1.25]; d=0; b=1;']);
   save_system('ex_han2');
```

同时还可以设置信号发生器的函数为方波，其频率为 0.1。这样就能给控制器设置所需的参数，对这样的系统进行仿真，就可以得出如图 6-19 (a) 所示的控制效果，在该图形中还同时绘制了输入信号。可以看出，这样控制器下的效果还是很理想的，它将很难控制的时变系统变得较易控制。图 6-19 (b) 中还给出了控制信号。

自抗扰控制器的特色是它不依赖受控对象模型，亦即该控制器有很强的鲁棒性。即使受控对象有很大的变化，整个控制效果都会保持得很好。考虑给定的受控对象模型，假设

$$\begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = 15\mathrm{Sat}(\cos t) + u(t) \end{cases}$$

其中 Sat(·) 为饱和函数，则可以搭建起如图所示 6-20 的系统框图，对其进行仿真则将得出如图 6-21 所示的输出曲线。从仿真结果可以看出，即使受控对象发生了很大的变化，最终的控制效果没有太大的改变。



(a) 输入和输出效果　　　　(b) 输入信号

图 6-19　时变系统控制效果

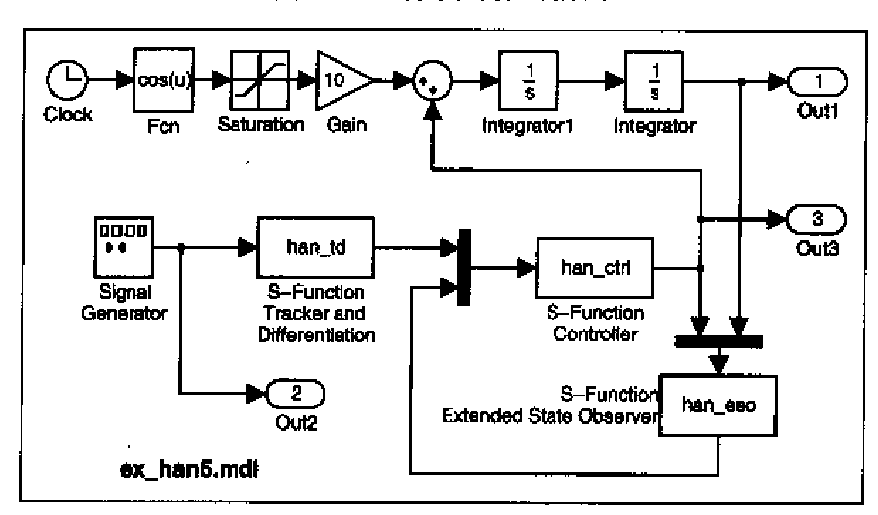


图 6-20　受控对象变化后的仿真模型

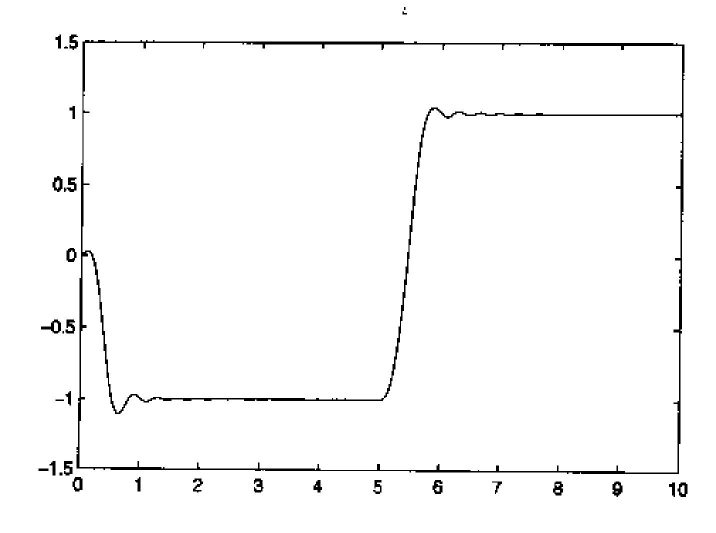


图 6-21　受控对象变化后的仿真效果

从上面的例子可以看出，由于采用 MATLAB/Simulink 定义、搭建模型是一个非常简单、直观的过程，所以可以容易地测试不同算法、不同受控对象下系统控制的效果。例如可以像前面叙述的那样容易地改变受控对象模型，立即就能得出控制的效果，这样做对系统分析和设计都是相当有效的。

6.3.3　用 C 语句编写 S-函数

除了用 MATLAB 语句来编写 S-函数外，还可以采用 C, C++, Fortran, Ada 等语言来编写 S-函数，例如，可以用 MATLAB 6.1 中提供的 S-函数编辑程序来设计 C 语言的 S-函数模板，该程序在 Functions & Tables 中的 S-Function Builder 模块给出，双击该模块将给出如图 6-22 所示的对话框，在该对话框中输入适当的参数则可以设计出 C 语言的 S-函数模板。在给出的界面中，可以输入有关初始化信息。然而，由于该程序刚刚推出，其功能极其简单，很不完备，尚未有实际应用的可能，在即将正式推出的MATLAB 6.5/Simulink 5.0 中该程序有所改观，但作者认为还是手写的方式更好些。

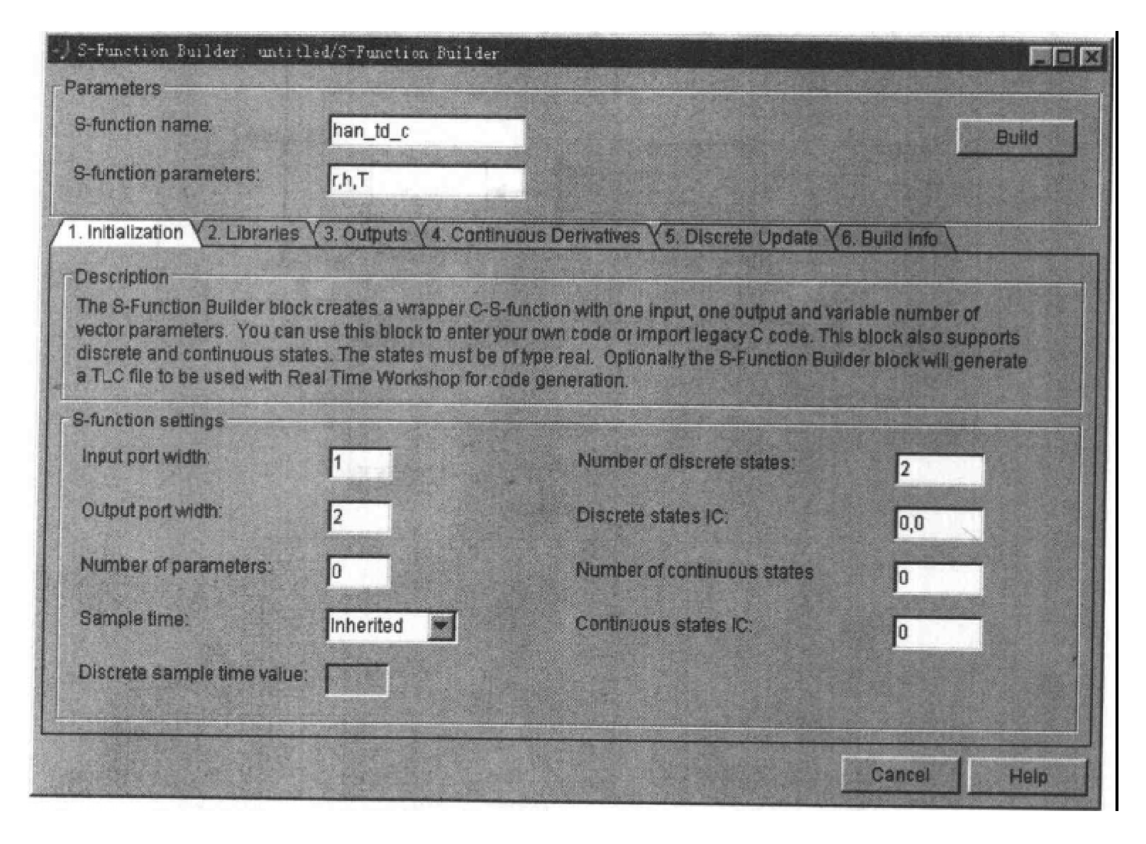


图 6-22　C 版本的 S-函数生成界面

Simulink 4.1 中提供的大量的 S-函数例子，打开 S-function demos 模块组，则可以发现有如图 6-23 所示的模块，可以分别研究提供的例子，了解用其他语言进行 S-函数进行编程的方法。

〖例 6.16〗 *再考虑式 (6.14) 中给出的跟踪-微分器算法，依据 Simulink 中提供的 C 语言模板程序，例如 MATLAB 目录中的* `simulink\src\sfuntmpl_basic.c` *文件，可以建立起如下所示的 C 语言*

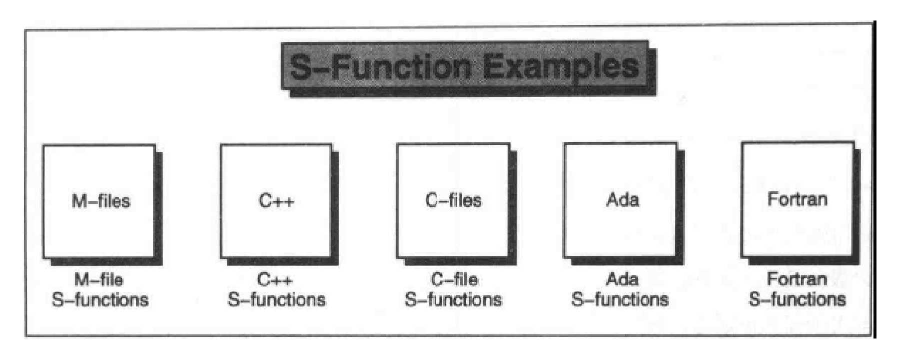


图 6-23 S-函数演示模块组

S-函数。

```
#define S_FUNCTION_NAME  sfun_han /* 应该写成实际的S-函数名 */
#define S_FUNCTION_LEVEL 2        /* 2 级 S-函数 */
#include "simstruc.h"
#include "math.h" /* 数学计算需要调用该库 */
double sign(double x) /* 自定义sign函数，即符号函数 */
{
   double f;
   f=1; if (x<=0) f=-1; return(f);
}
double fst(real_T *x, const real_T *u, const real_T *p1, const real_T *p2)
{  /* FST函数，由于算法简单，不再详细解释 */
   double delta, delta0, a0, a, y, r, h;
   r=p1[0]; h=p2[0];
   delta=r*h; delta0=delta*h;
   y=x[0]-u[0]+h*x[1];
   a0=sqrt(delta*delta+8.0*r*fabs(y)); /* 注意，浮点绝对值函数为fabs */
   if (fabs(y) <= delta0) a=x[1]+y/h;
   else a=x[1]+0.5*(a0-delta)*sign(y);
   if (fabs(a) <= delta) return (-r*a/delta);
   else return(-r*sign(a));
   }
static void mdlInitializeSizes(SimStruct *S) /* 初始化函数 */
{
    ssSetNumSFcnParams(S, 3);  /* 附加参数个数 */
    ssSetNumContStates(S, 0);  /* 连续状态个数 */
    ssSetNumDiscStates(S, 2);  /* 离散状态个数 */
    if (!ssSetNumInputPorts(S, 1)) return;
```

```
    ssSetInputPortWidth(S, 0, 1); /* 输入信号设定 */
    ssSetInputPortDirectFeedThrough(S, 0, 0); /* 是否将输入直接传到输出 */
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 2);  /* 输出信号设置为2路 */
    ssSetNumSampleTimes(S, 1);      /* 采用周期个数 */
    ssSetNumRWork(S, 3);            /* 3个附加参数情况 */
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, 0);
}
static void mdlInitializeSampleTimes(SimStruct *S)
{   /* 采样周期设置子程序 */
    ssSetSampleTime(S,0,*mxGetPr(ssGetSFcnParam(S,2))); /*等于第3附加参数*/
    ssSetOffsetTime(S, 0, 0.0);
}
static void mdlOutputs(SimStruct *S, int_T tid)
{   /* 系统输出方程的写法 */
    const real_T *x = ssGetRealDiscStates(S); /*用这样固定方式获得x,y指针*/
    real_T       *y = ssGetOutputPortSignal(S,0);
    y[0] = x[0]; y[1] = x[1];   /* 写出输出方程 */
}
static void mdlUpdate(SimStruct *S, int_T tid)
{  /* 状态更新式子 */
    real_T       *x = ssGetRealDiscStates(S);
    const real_T *u = (const real_T*) ssGetInputPortSignal(S,0);
    const real_T *r = mxGetPr(ssGetSFcnParam(S,0)); /* 获得各个附加参数 */
    const real_T *h = mxGetPr(ssGetSFcnParam(S,1));
    const real_T *T = mxGetPr(ssGetSFcnParam(S,2));
    real_T tempX[2] = {0.0, 0.0}; /* 注意这里要引入应该暂存的数组 */
    tempX[0] = x[0] + T[0]*x[1];  /* 不能直接用x[0]=...语句 */
    tempX[1] = x[1] + T[0]*fst(x,u,r,h);
    x[0] = tempX[0]; x[1] = tempX[1]; /* 将暂存数组的内容赋给x */
}
```

由于篇幅所限，将冗长的注释语句及若干空白函数略去，将必要的修改部分用中文注释给出，但应该注意，用 MATLAB 提供的 LCC 编译器不支持中文注释，具体应该察看相关的源程序。

编写了 C 语言程序后，还需要对之进行编译，生成所需的动态连接库文件 (DLL 文件)，第一次运行 C 语言编译器前需要进行编译环境设置．在 MATLAB 的命令窗口中给出下面的命令

```
>> mex -setup
```

按照提示回答一系列问题，就可以建立起和一个 C 编译器之间的关系，用户可以根据需要选择 MATLAB 自带的 LCC 编译器或机器上安装的 Visual C++ 编译器。

建立起和 C 语言编译器之间的关系，则需要给出下面的命令对 C 程序进行编译

```
>> mex sfun_han.c
```

注意，在编译时一定要给出后缀名。如果程序本身没有错误，则将生成 sfun_han.dll 文件，该文件的作用和前面建立的 han_td.m 同样使用。

用同样的方法建立起扩张的状态观测器文件 sfun_eso.c 和控制器文件 sfun_ctr.c，限于篇幅，不能给出这些函数清单，但这些文件同样可以下载。由这两个 C 文件可以建立起动态链接连接库文件，将建立起如图 6-24 所示的控制系统框图，事实上，因为用 C 语言编写的 S-文件的接口与以前的 MATLAB S-函数完全一致，只需用新的 S-函数名替换图 6-18 中的即可。

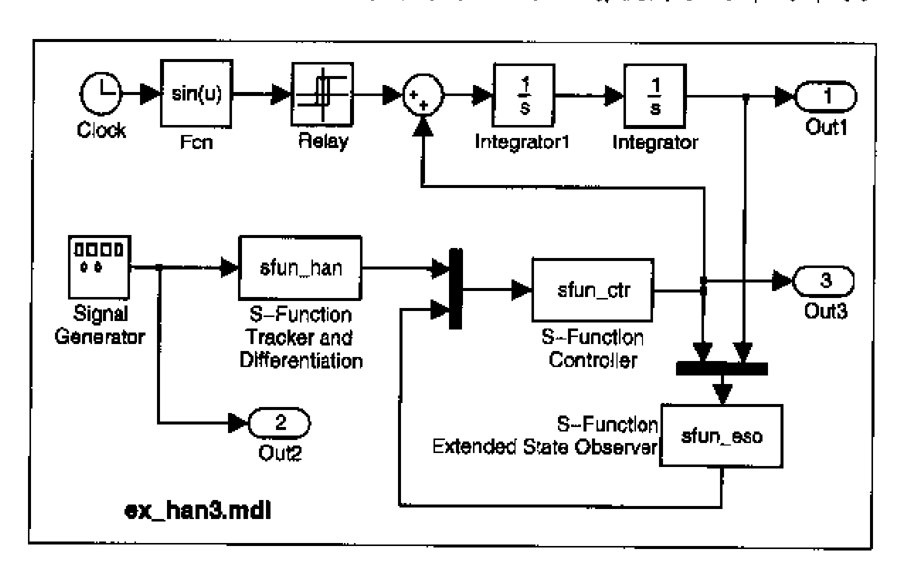


图 6-24 自抗扰控制器的仿真模型 (C 版 S-函数)

比较两个 Simulink 模型，可以发现

```
>> tic, [t,y]=sim('ex_han2'); toc % MATLAB 版 S-函数
elapsed_time =
    3.5100
>> tic, [t,y]=sim('ex_han3'); toc % C 版 S-函数
elapsed_time =
    0.2800
```

可见，用 C 语言的 S-函数执行起来速度比相应的 MATLAB 语言 S-函数快 10 多倍。

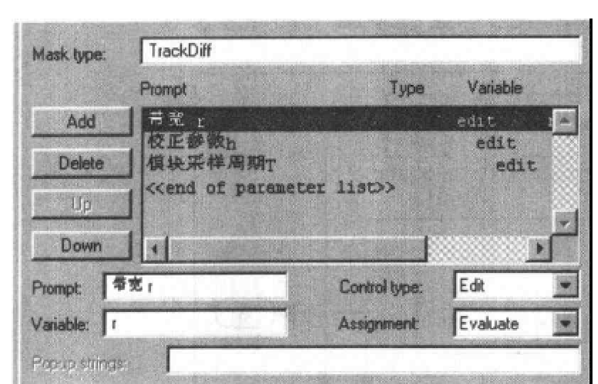
由前面的例子可以看出，编写和调试 C 格式的 S-函数比编写同样的 MATLAB 格式的 S-函数复杂得多，所以在纯仿真中最好使用 MATLAB 格式去编写。但在一些应用中，由于 MATLAB 格式的 S-函数不能转换成 C 语言程序，生成独立文件，所以应该采用 C

语言去编写 S-函数。

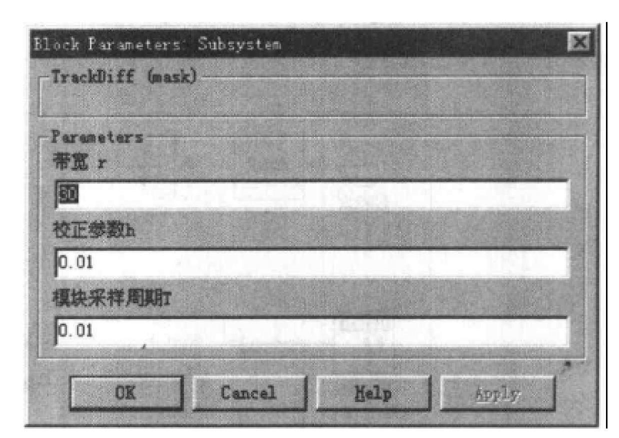
6.3.4 S-函数模块的进一步改进

我们知道，依照前面方法设计出的 S-函数虽然可以带有附加参数，但模型参数输入较不方便，因为需要在附加参数栏目中同时输入若干个参数，没有任何提示，所以可以用封装模块的方法来封装该 S-函数，设计出相应的参数输入对话框。

〖例 6.17〗 仍以跟踪-微分器为例来演示参数输入对话框的设计。首先按照图 6-16 (b) 所示的格式先建立起跟踪-微分器模型，右击该模块可以获得快捷菜单，选中菜单项 Edit | Mask S-function 菜单项封装该子系统，得出一个参数输入对话框，其设置部分如图 6-25 (a) 所示，用这样的方法可以将所需的参数设置提示。



(a) 封装模块对话框内容



(b) 模块参数输入对话框

图 6-25 封装的跟踪-微分器模块

构造出封装模块后，双击之，则能得出如图 6-25 (b) 所示的对话框，可以看出，该对话框允许以更容易的方式输入附加参数，因为每个参数都给出一个提示。

6.4 Stateflow 原理与使用技巧

Stateflow 是有限状态机的图形实现工具，它可以用于解决复杂的监控逻辑问题，用户可以用图形化的工具来实现各个状态之间的转换。Stateflow 生成的监控逻辑可以直接嵌入到 Simulink 模型下，从而实现二者的无缝连接。事实上，在仿真初始化过程中，Simulink 将自动启动编译程序，将 Stateflow 绘制的逻辑框图变换为 C 格式的 S-函数，从而在仿真过程中直接调用相应的动态连接库文件，将二者构成一个仿真整体的。

6.4.1 有限状态机简介

Stateflow 仿真的原理是有限状态机 (finite state machine，简称 FSM) 理论，所谓有限状态机，就是指在系统中有可数的状态，在某些事件发生时，系统从一个状态转换成

另一个状态，所以有限状态机系统又称为事件驱动的系统。在有限状态机的描述中，可以设计出从一个状态到另一个状态转换的条件，在每对相互可转换的状态下都设计出状态迁移的事件，从而构造出状态迁移图。

在 Stateflow 中提供了图形界面支持的设计有限状态机的方法，它允许用户建立起有限的状态，并用图形的形式绘制出状态迁移的条件，从而构造出整个有限状态机系统。所以在 Stateflow 下，状态和状态转换是其最基本的元素，有限状态机的示意图如图 6-26 所示。所谓有限，是指其中的状态或模态的个数是可数的，故而能用这样的示意图表示出来。在本示意图中有 4 个状态，这几个状态直接的转换是有条件的，其中有的是状态之间相互转换的，还有 A 状态自行转换的。在有限状态机的表示中，还应该表明这些状态迁移的条件或事件。

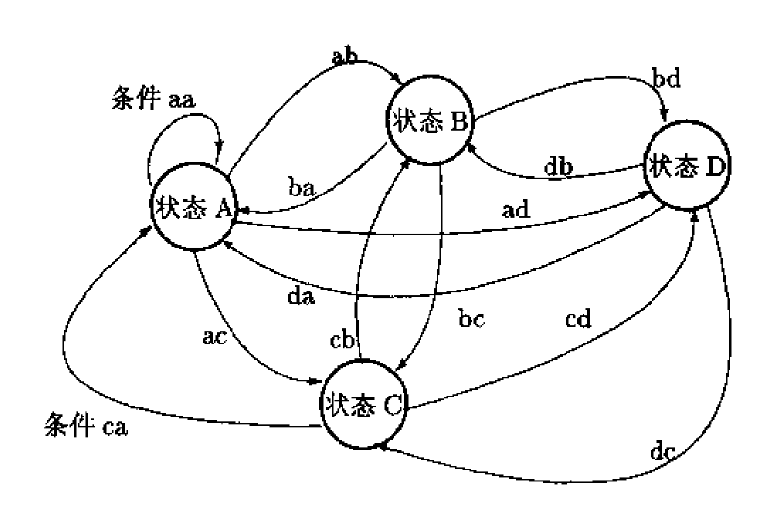


图 6-26 有限状态机示意图

Stateflow 模型一般是嵌在 Simulink 模型下运行的，Stateflow 图是事件驱动的，这些事件可以在同一 Stateflow 图中，也可能来自 Simulink。

6.4.2 Stateflow 应用基础

在 MATLAB 的命令窗口提示符下键入 `stateflow` 命令，将打开如图 6-27 所示的界面，其中左边的窗口，右边的窗口中为 Simulink 窗口，其中 Charts 为空白的 Stateflow 模块图标，在 Simulink 模型中可以直接嵌入 Stateflow 模块。

双击其中的 Stateflow 模块，将得出如图 6-28 所示的编辑界面，用户可以在此窗口中编辑所需的 Stateflow 模型。Stateflow 提供了强大的框图编辑功能，可以描述很复杂的逻辑关系式。应该指出的是，Stateflow 框图输出的应该为状态，且这些状态的值是可枚举的。

在 Stateflow 编辑界面内右击鼠标，则将得出如图 6-29 (a) 所示的快捷菜单，选择其中的 Properties (属性) 菜单，则将得出如图 6-29 (b) 所示的对话框，用户可以从中设置整个模型的属性。

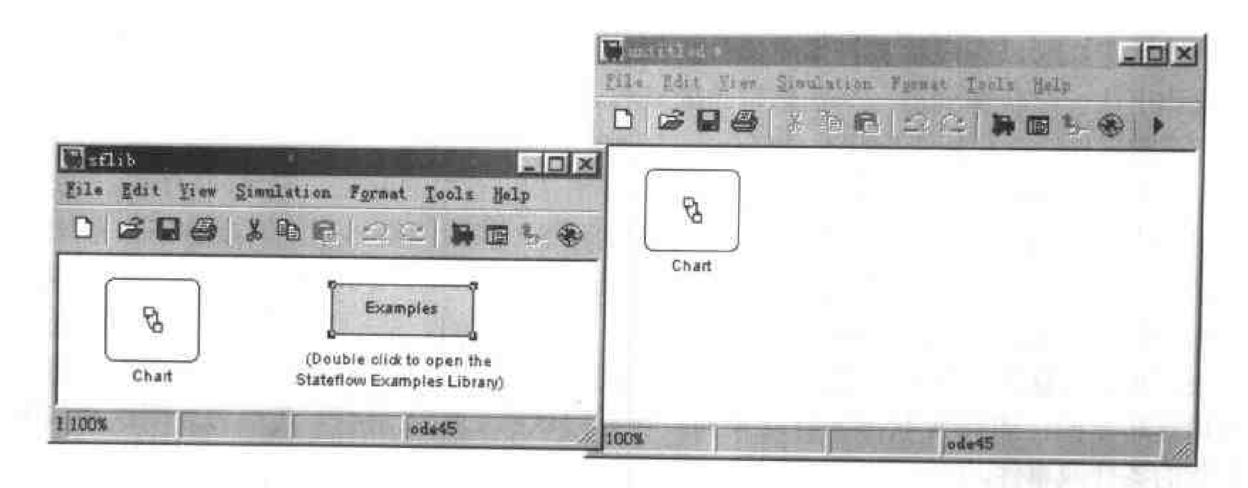


图 6-27 Stateflow 启动界面

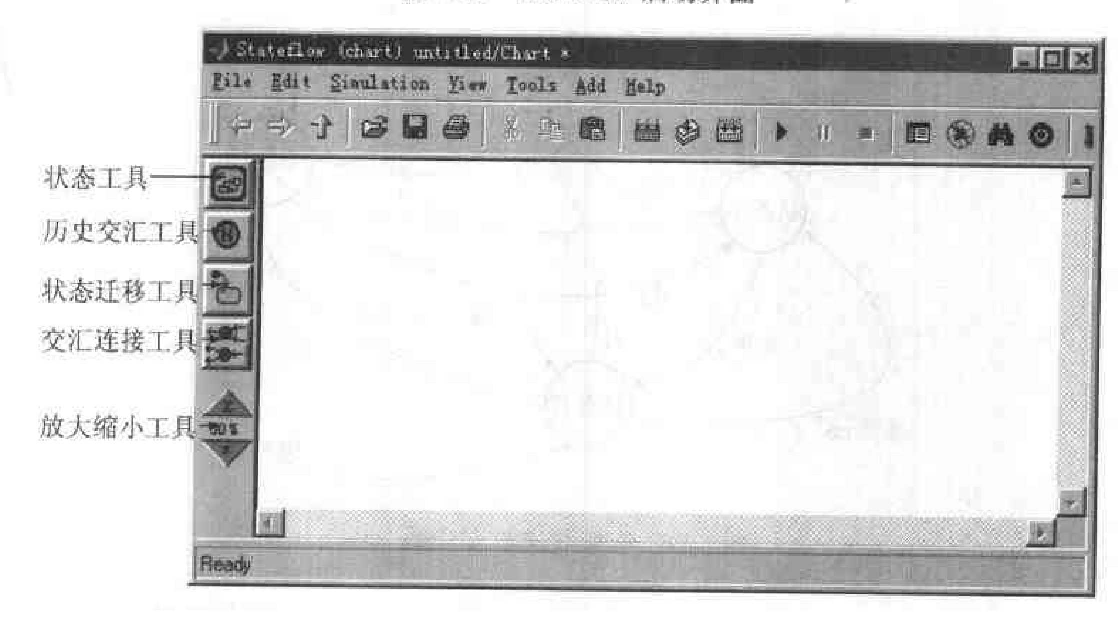
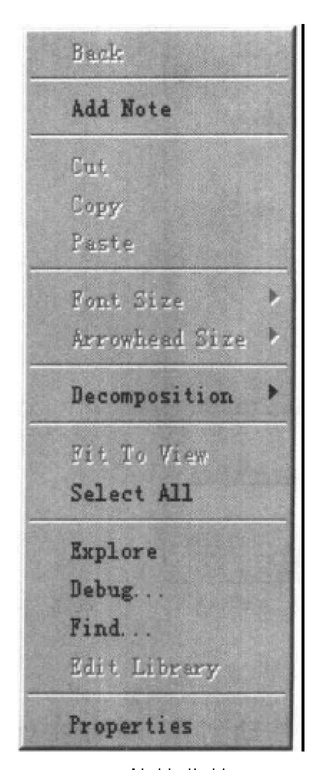


图 6-28 Stateflow 图形编辑程序界面

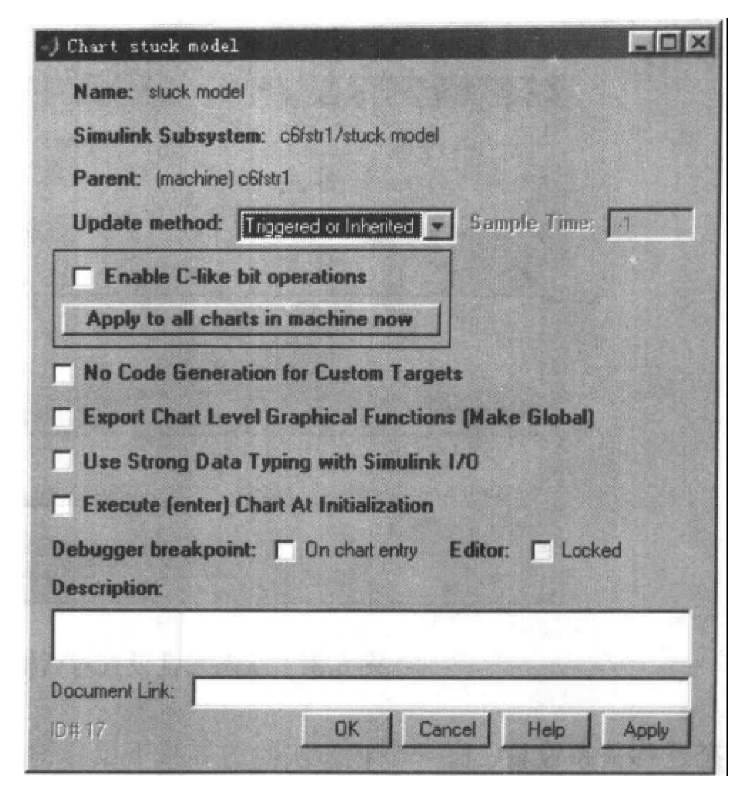
在该 Stateflow 模型编辑界面的左侧有四个编辑工具，可以利用这些编辑工具绘制出 Stateflow 图形。下面将分别介绍这些编辑工具：

- **状态工具** 系统的状态就是系统运行的模态。在 Stateflow 下，状态有两种行为，即“活动的”(active) 和“非活动的”(inactive)。状态工具允许用户在模型中添加一个状态，单击该按钮，则可以在图形编辑窗口中绘制一个状态的示意模块，该状态左上角标注一个问号，用户可以在该问号的位置填写有关状态的名称和动作描述，如使用名称 on。使用该工具，可以绘制出所有需要的状态，如图 6-30 所示。

  如果右击建立的状态图标，并选择快捷菜单中的 Properties 菜单项，则将得出如图 6-31 所示的对话框，可以在此对话框中输入状态的属性。双击该图标也将进入状态模块的编辑状态。例如，可以在 Label (标示) 栏目填写上所需的标示和状态的赋值内容，其具体格式如：



(a) 快捷菜单



(b) 属性设置对话框

图 6-29 Stateflow 整体设置

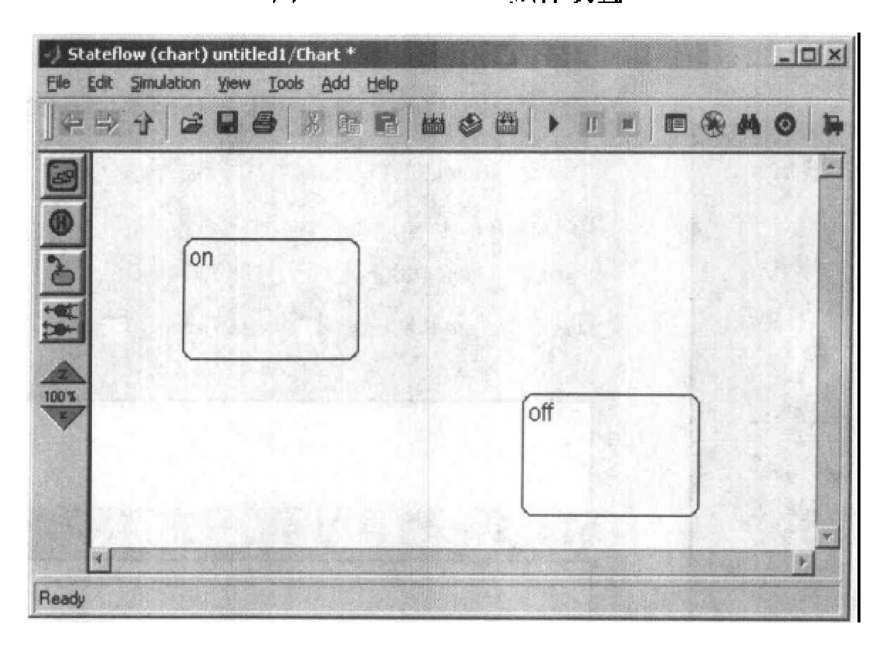


图 6-30 Stateflow 窗口的新建状态

```
stuck/
entry: stuck=1;
```

其中 sliding 为模块名称，entry: 语句为状态的赋值，亦即将状态 stuck 的值赋为

1，注意这里应该使用 C 语言格式进行变量赋值。

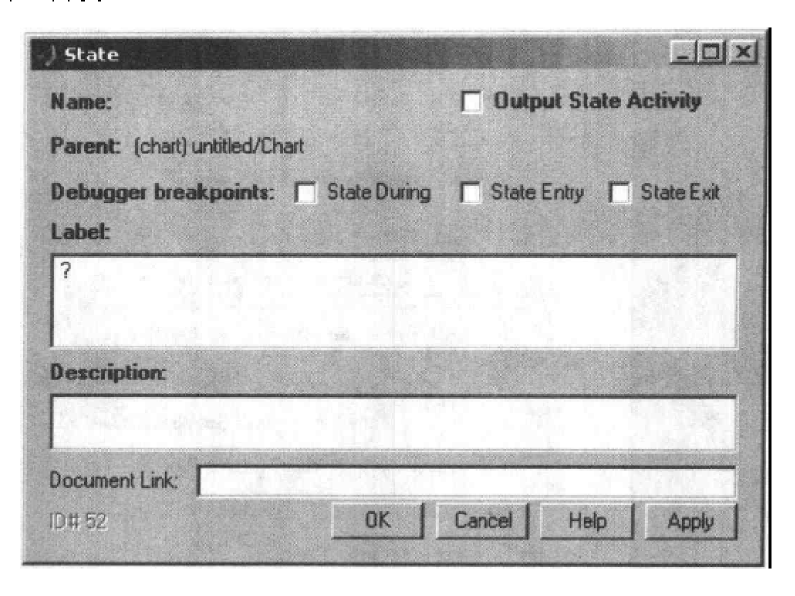
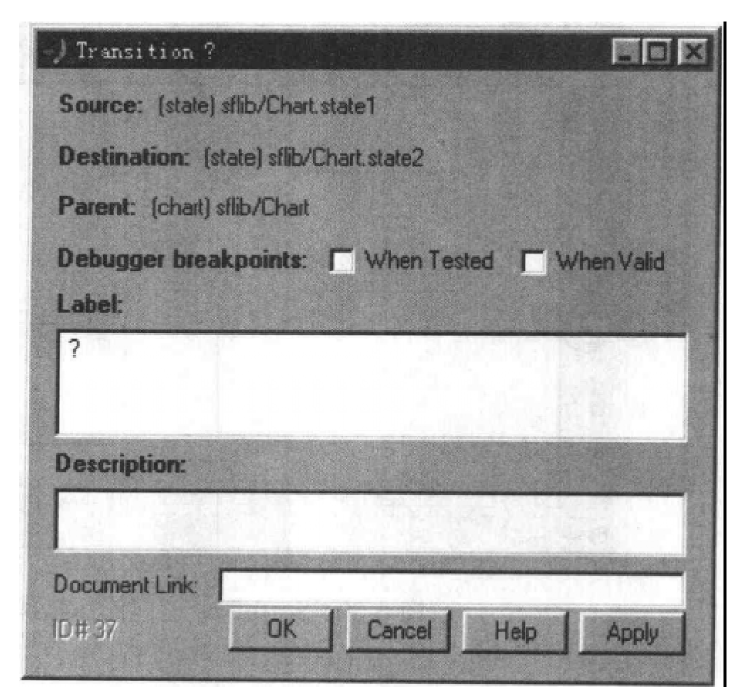


图 6-31　状态属性设置对话框

- **状态迁移关系设置**　在一个状态块的边界按下鼠标键，并拖动到另一个状态处释放，则可以绘制出从一个状态转换成另一个状态的连线。右击这条状态转换线，则将得出如图 6-32 (a) 所示的快捷菜单，选择该菜单中的 Properties 菜单项，则将得出如图 6-32 (b) 所示的对话框，在对话框的 Label 栏目中可以添加状态迁移的关系式。



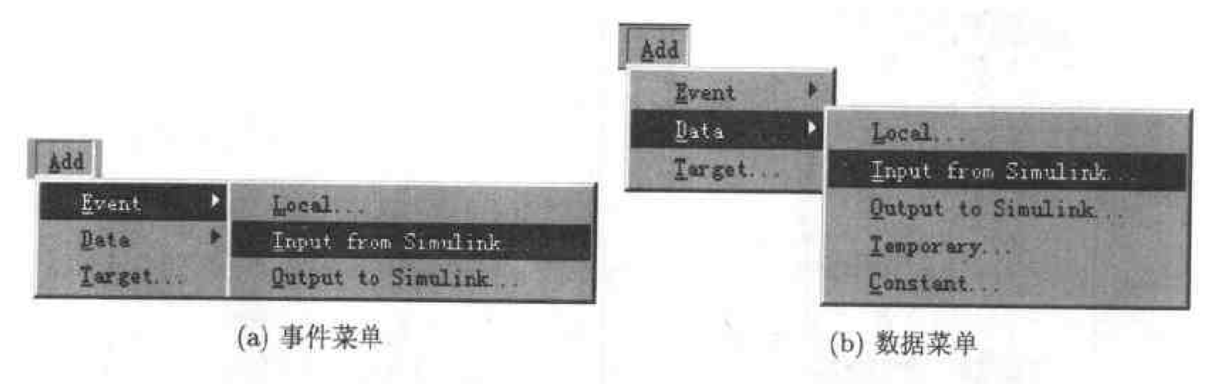
(a) 快捷菜单



(b) 属性设置对话框

图 6-32　Stateflow 状态迁移设置

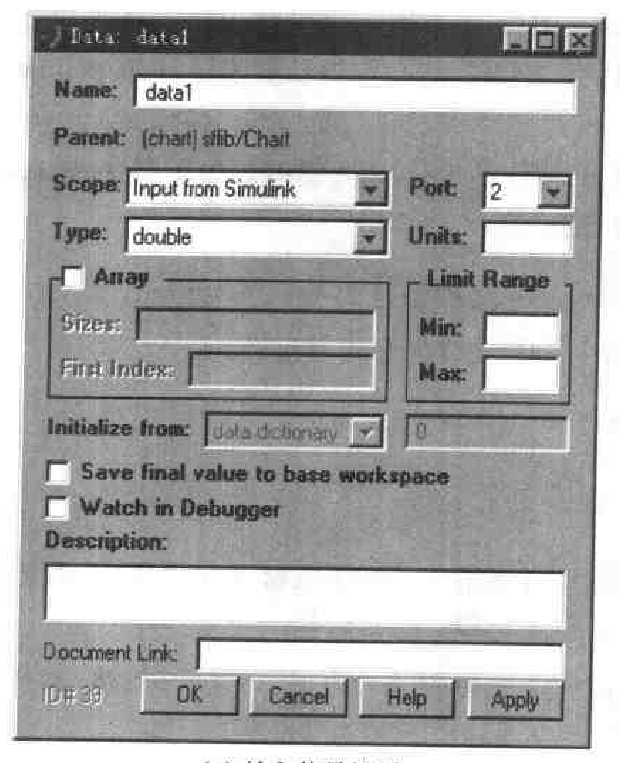
- **事件与数据设置** Stateflow 提供了一个 Add 菜单，其内容如图 6-33 (a) 所示。用户可以从中选择适当的事件与数据定义，例如用户可以从 Simulink 输入事件。Stateflow 框图还允许用户与 Simulink 环境交互数据，这时应该选择如图 6-33 (b) 所示菜单项。

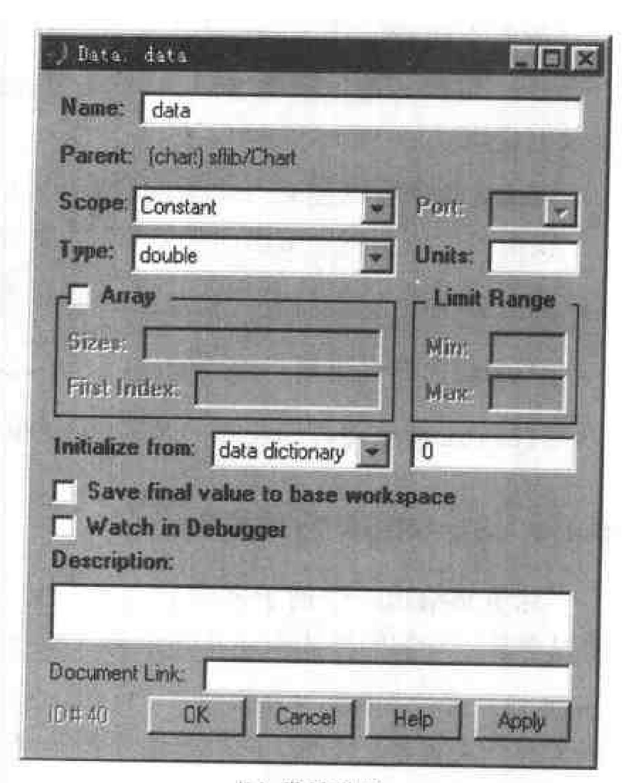


(a) 事件菜单 (b) 数据菜单

图 6-33 Add 菜单内容

- **输入输出设置** 选择 Add | Data | Input from Simulink 菜单项则可以得出如图 6-34 (a) 所示的对话框，用户可以给该变量设置变量名，数据类型等消息。这样的设置将给 Stateflow 模块添加输入端子。





(a) 输入信号设置 (b) 常量设置

图 6-34 输入循环与常量设置对话框

如果选择 Add | Data | Constant 菜单项，则可以得出如图 6-34 (b) 所示的对话框，用户可以给该模块所使用的变量以常数的形式赋值。输出变量也可以用相应的

方法进行设置。设置了变量后，可以选择 Tools | Explore 菜单项对各个信号进行显示，这样将得出如图 6-35 所示的对话框，可以轻易地修改输入变量的名称、类型和顺序。

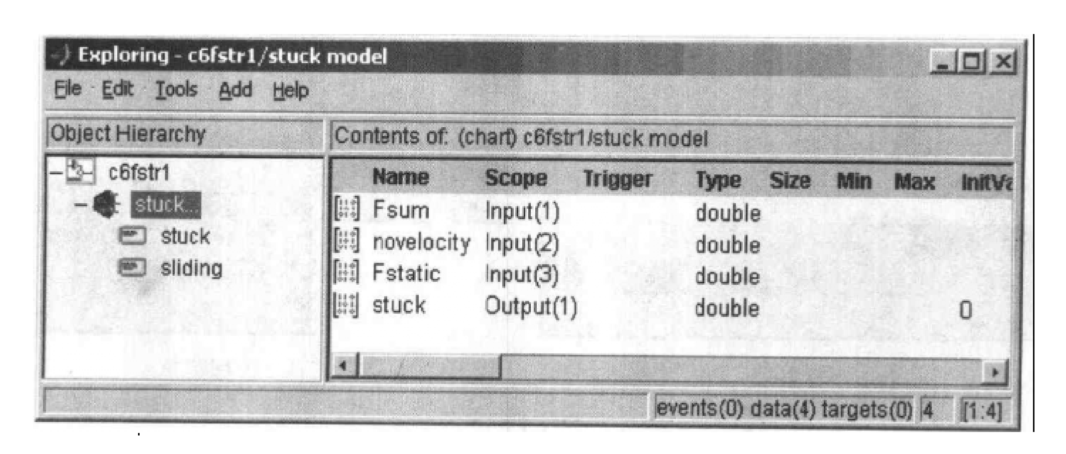


图 6-35　Stateflow 内容分析界面

6.4.3　Stateflow 的常用命令

前面介绍过，`stateflow` 命令可以打开 Stateflow 主界面，可以使用该命令来启动 Stateflow 编辑环境，开始 Stateflow 流图的绘制。还有一些命令可以用来进行 Stateflow 工具的使用：

- `sfnew` 命令可以创立一个新的带有 Stateflow 模块的 Simulink 模型。如果用“`sfnew` 模型名”语句将用给出的模型名建立起这样的模型。
- `sfexit` 命令将关闭所有包含 Stateflow 建模的窗口，并退出 Stateflow 环境。
- `sfsave` 命令允许用户将绘制出的 Stateflow 模型用文件的形式存起来。
- `sfprint` 命令将打印绘制的 Stateflow 模型。

6.4.4　Stateflow 应用举例

这里将给出一个例子来演示 Stateflow 的建模与应用。该例子取材于文献 [32]，我们将以更易于理解的格式叙述其理论背景和整个系统的建模、仿真过程。

【例 6.18】 考虑图 6-36 中给出的力学模型，在该模型中，重物 M 在外力 $F_{\rm in}$ 的作用下运动。这样物体 M 所受的力包括外力 $F_{\rm in}$，摩擦力 $F_{\rm f}$ 和弹簧的拉力 $F_{\rm str}$。可以得出合力为

$$F = F_{\rm in} - F_{\rm f} - F_{\rm s} \tag{6.21}$$

根据 Newton 第二运动定律，可以立即写出下面的方程

$$M\ddot{x} = F = F_{\rm in} - F_{\rm f} - F_{\rm str} \tag{6.22}$$

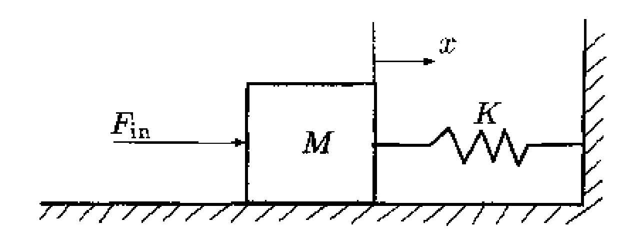


图 6-36 力学模型的示意图

其中 x 为位移，所以 $\ddot{x}$ 为加速度。弹簧的拉力 $F_{\text{str}} = Kx$, K 倔强系数，摩擦力 F_{f} 可以由下面的公式计算出来

$$F_{\text{f}} = \begin{cases} \text{sign}(\dot{x})\mu F_n, & |F_{\text{sum}}| > \mu F_n \\ F_{\text{sum}}, & \dot{x} = 0 \text{ 且 } |F_{\text{sum}}| \leq \mu F_n \end{cases} \tag{6.23}$$

其中 μ 为摩擦系数，F_n 为正压力 (法向力)，$\dot{x}$ 为物体的速度，当 $\dot{x} = 0$ 表示物体静止。F_{sum} 为静止状态下的受力，满足 $F_{\text{sum}} = F_{\text{in}} - F_{\text{sliding}}$。当速度为非零时，将需要一个冲力使之瞬时回零，该冲力往往超过最大的允许限度 F_{sliding}。当物体的速度已经为零，则 F_{sum} 力将维持该物体的加速度为零。摩擦力一般可以再分为静摩擦力和动摩擦力，这主要取决于物体是否运动而言，亦即

$$\mu F_{\text{n}} = \begin{cases} \mu_{\text{static}} F_{\text{n}} = F_{\text{static}}, & \dot{x} = 0 \\ \mu_{\text{sliding}} F_{\text{n}} = F_{\text{sliding}}, & \dot{x} \neq 0 \end{cases} \tag{6.24}$$

其中 μ_{static} 和 μ_{sliding} 分别为静脉摩擦系数和滑动摩擦系数，综合考虑上述两个条件式，则可以由下面的式子求出摩擦力

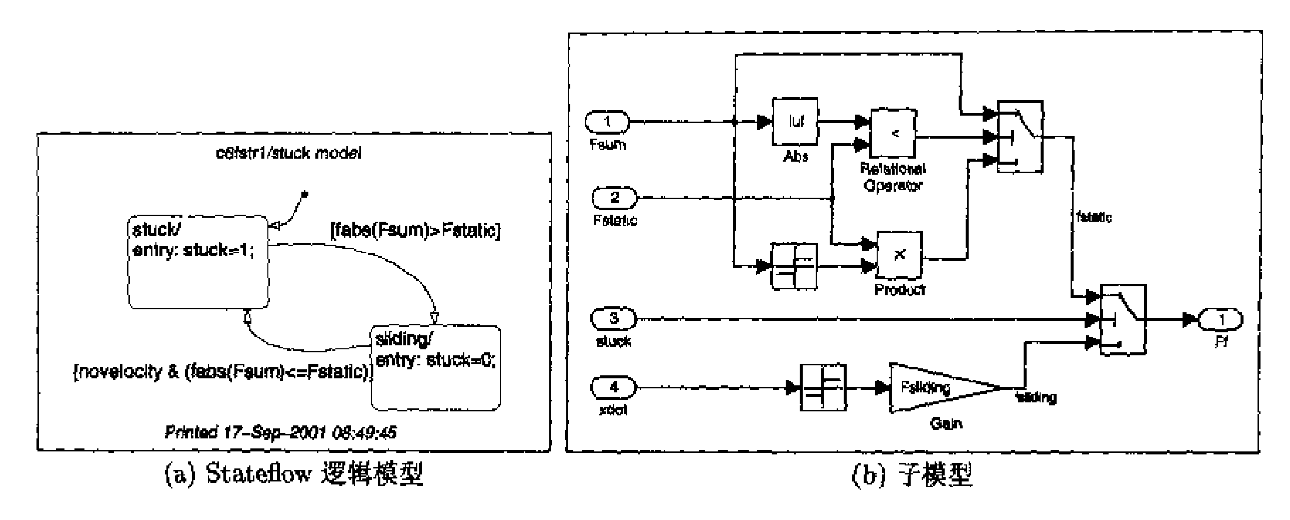
$$F_{\text{f}} = \begin{cases} \text{sign}(\dot{x})F_{\text{sliding}}, & \dot{x} \neq 0 \\ F_{\text{sum}}, & \dot{x} = 0 \text{ 且 } |F_{\text{sum}}| < F_{\text{static}} \\ \text{sign}(F_{\text{sum}})F_{\text{static}}, & \dot{x} = 0 \text{ 且 } |F_{\text{sum}}| \geq F_{\text{static}} \end{cases} \tag{6.25}$$

从式 (6.23) 中可以看出，在 $|F_{\text{sum}}| > F_{\text{static}}$ 时，系统处于运动状态，故这时设置 stuck 标志为 0，若上述条件不满足，且系统处于静止状态，可以令 stuck 标志为 1，通过这样的方法设置可以设置摩擦的状态。根据这样的逻辑描述，可以构造出一个 Stateflow 框图，如图 6-37 (a) 所示。在该 Stateflow 框图下，定义合力 `Fsum`，零速度检测信号 `novelocity` 和静态摩擦力 `Fstatic` 为来自 Simulink 的输入信号，再令标志状态 stuck 为输出到 Simulink 的信号。

另外，根据式 (6.25)，可以建立起摩擦力 `Ff` 的子系统模型，如图 6-37 (b) 所示。在该系统中使用了两个开关模块来描述三个条件。

有了这两个模块，就可以最终搭建出整个系统的 Simulink 模型，如图 6-38 所示。在此模型中，用了 Hit Crossing 模块判定速度是否为零。另外采用了双积分器的方式来描述式 (6.22) 中表示的数学模型。注意，这样绘制的模型和文献 [32] 中给出的不完全一致，从结构来说更易于理解。

在 Simulink 模型中，速度信号不是取自第一积分器的输出，而是取自其状态，这就需要将该积分器的 Show state port 复选框选中，这样做是为了避免代数环的现象。另外需要用 stuck 信号来复位该积分器。



(a) Stateflow 逻辑模型　　　　(b) 子模型

图 6-37　摩擦系统子系统模型

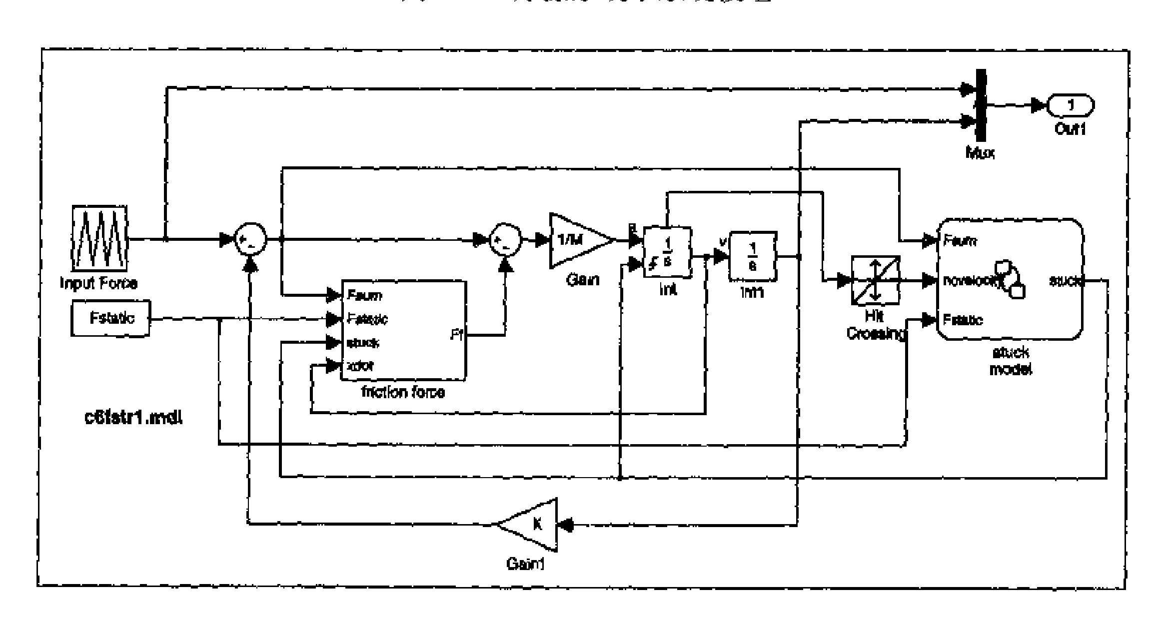


图 6-38　摩擦系统的 Simulink 模型

在仿真系统中，可以选择三角波信号作为系统的输入，即外力。选择输入源模块组中的 Repeating Sequence 图标，双击该图标则可以得出如图 6-39 所示的参数输入对话框。该参数组的含义是生成周期为 10 的重复信号，使得时间等于 5 时达到其峰值 5。

由于该系统属于刚性系统，所以在选择仿真算法时应该选择 ode15s，并将绝对误差限设置为 1e-6，这样就可以较可靠地对原系统进行仿真了。

建立起系统的仿真模型后，可以用下面的语句将有关参数输入到 MATLAB 的工作空间

```
>> M=0.01; K=1; Fsliding=1; Fstatic=1;
```

这时启动仿真过程将在命令窗口中给出下面的提示

```
Parsing successful for machine: "c6fstr1"(#30)  % 解析过程
Code Directory :                                % 代码翻译目录
     "C:\matlab6p1\work\sfprj\build\c6fstr1\sfun\src"
```

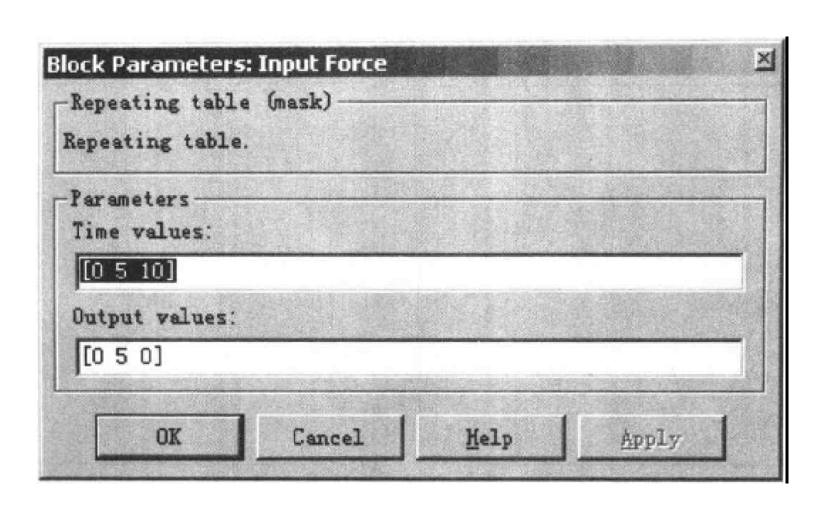


图 6-39 输入信号参数设置对话框

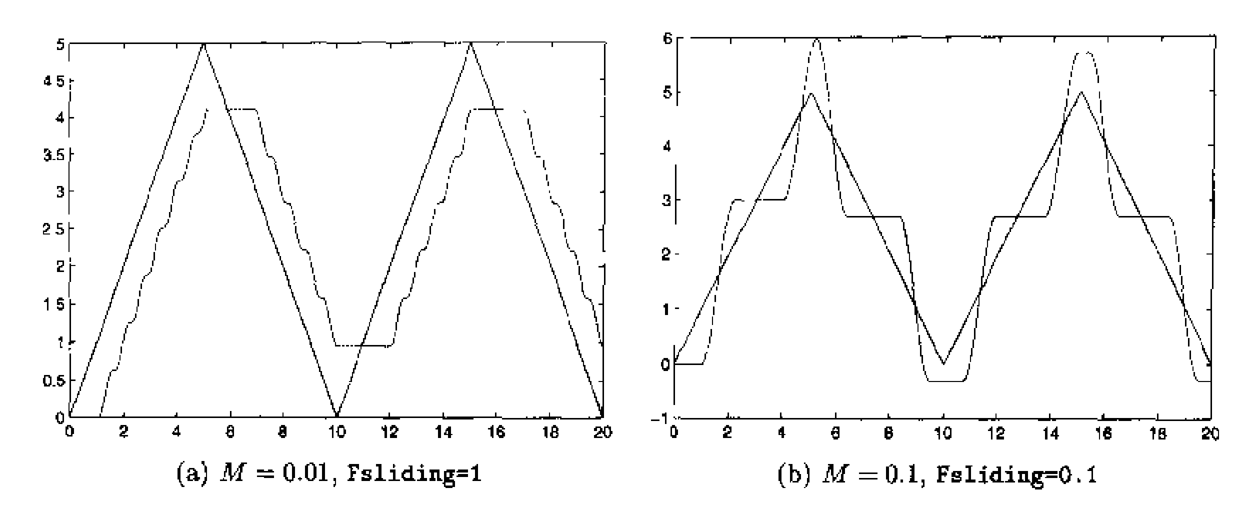
```
Machine (#30): "c6fstr1"  Target (#42): "sfun"  % 代码翻译
     "c6fstr1_sfun_debug_macros.h"
     "c6fstr1_sfun.h"
     "c6fstr1_sfun.c"
Chart "stuck model Stateflow" (#31):              % 自动机代码
     "c6fstr1_sfun_c1.h"
     "c6fstr1_sfun_c1.c"
Interface and Support files:                      % 注册等支持信息生成
     "c6fstr1_sfun_registry.c"
     "c6fstr1_sfun.bat"
     "c6fstr1_sfun.lmk"
     "c6fstr1_sfun.lmko"
Code generation successful for machine: "c6fstr1"  % 可执行代码
Stateflow Making simulation target "c6fstr1_sfun", ...% Stateflow 可执行代码
% 中间若干条代码连接命令从略
Time: 2.213 seconds                                % 整个过程所需时间
Make successful for machine: "c6fstr1"             % 成功完成状态机的构造
```

表明在仿真之前，需要将Stateflow模块翻译成C语言的S-函数程序，再通过MATLAB提供的编译器将之编译成动态连接库 (DLL) 文件 c6fstr1.dll，这样才能正常嵌入Simulink环境进行仿真。

自动生成的C语言源程序包括 c6fstr1_sfun.c, c6fstr1_sfun_c1.c, c6fstr1_sfun_registry.c，另外还有相应的头文件 (*.h)，编译连接的批处理文件 c6fstr1_sfun.bat。这里生成的C格式的S-函数过于烦琐，不建议也不适于手工修改和维护，所以在此不介绍其内部的语句结构。

由前面给出的系统参数取值，可以绘制出输入信号和输出位移信号的曲线，如图 6-40 (a) 所示，其中三角波为输入信号。

还可以改变原始的系统模型参数，例如将其中的参数修改为



(a) $M = 0.01$, Fsliding=1　　(b) $M = 0.1$, Fsliding=0.1

图 6-40　不同条件下输入、输出曲线

```
>> M=0.1; Fsliding=0.1;
```

再进行仿真，则将得出如图 6-40 (b) 所示的曲线。

在仿真过程中如果打开 Stateflow 窗口，则可以看出状态切换的过程，其中当前状态在仿真过程中颜色总在闪烁，说明内部在进行状态的切换。

6.4.5 控制流程的 Simulink 仿真模块

在 Simulink 4.1 (MATLAB 6.1) 版本中，流程控制子系统在 Subsystem 模块组中给出，它允许用户搭建表示条件转移、循环结构、开关结构和试探结构的 Simulink 框图。有些用 Stateflow 表示的简单的框图也可以用这样的流程模块表示。这里将以简单的例子介绍流程模块的使用。

【例 6.19】 例 6.18 中介绍的使用 Stateflow 表示逻辑关系的模块可以由 If 块取代。在 Subsystem 模块组中，有 If 模块和 Action Port 模块可以使用，具体使用方法是，将 If 模块复制到子系统窗口中，然后将 Action Port 模块连接到 If 模块的输出端，表示条件满足时执行的动作。

按照这样的思路，可以构造出如图 6-41 所示的子系统。和前面的 Stateflow 框图一样，该子系统有三个输入端子，Fsum, novelocity, Fstatic，分别用于构造转移的条件表达式。一个输出端子，表示 stuck 信号。

双击 If 模块，则得出如图 6-42 (a) 所示的对话框，在该对话框下可以输入转移条件的表达式。 例如在这里分别将 |Fsum|, Fstatic, novelocity 三个信号设置为 u1, u2 和 u3，故将其 Number of inputs (输入路数) 设置为 3，在 If expression 栏目下输入 u1>u2，表示 $|F_{\rm sum}| > F_{\rm static}$ 关系式，若此条件不满足，则相当于 $|F_{\rm sum}| \le F_{\rm static}$ 条件满足，在此条件下若满足 u3==0，则式 (6.25) 中的第二个条件式满足，故可以将该条件填写到 Elseif expression 栏目，这样就可以按要求设置出所需的 If 模块。可以看出，该模块有三个输出端子，第 1 个对应于 If expression，第 2 个对应于 Elseif expression，第 3 个对应于 Else 模块。

条件模块建立起来之后，需要为之配备执行子系统模块，可以将两个 Action Port 模块直接连

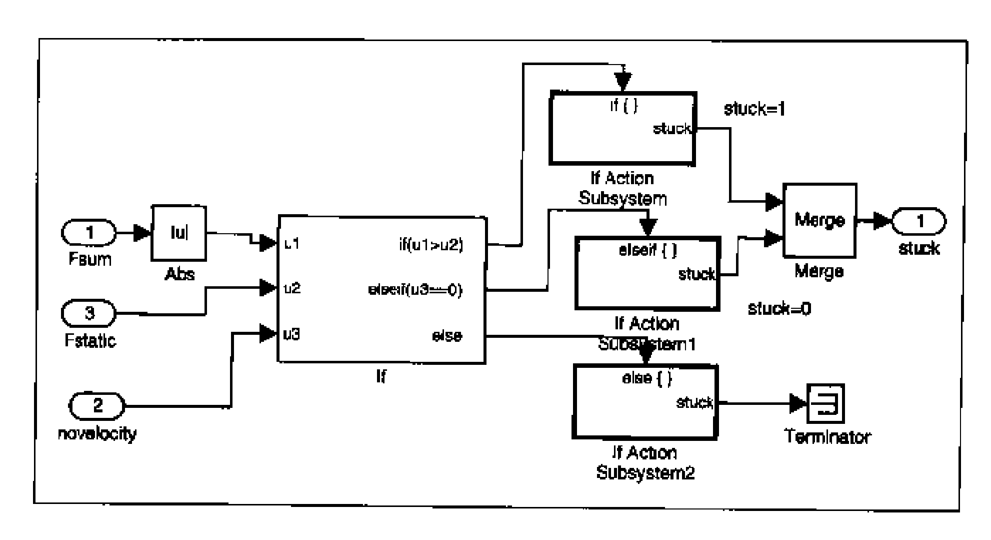
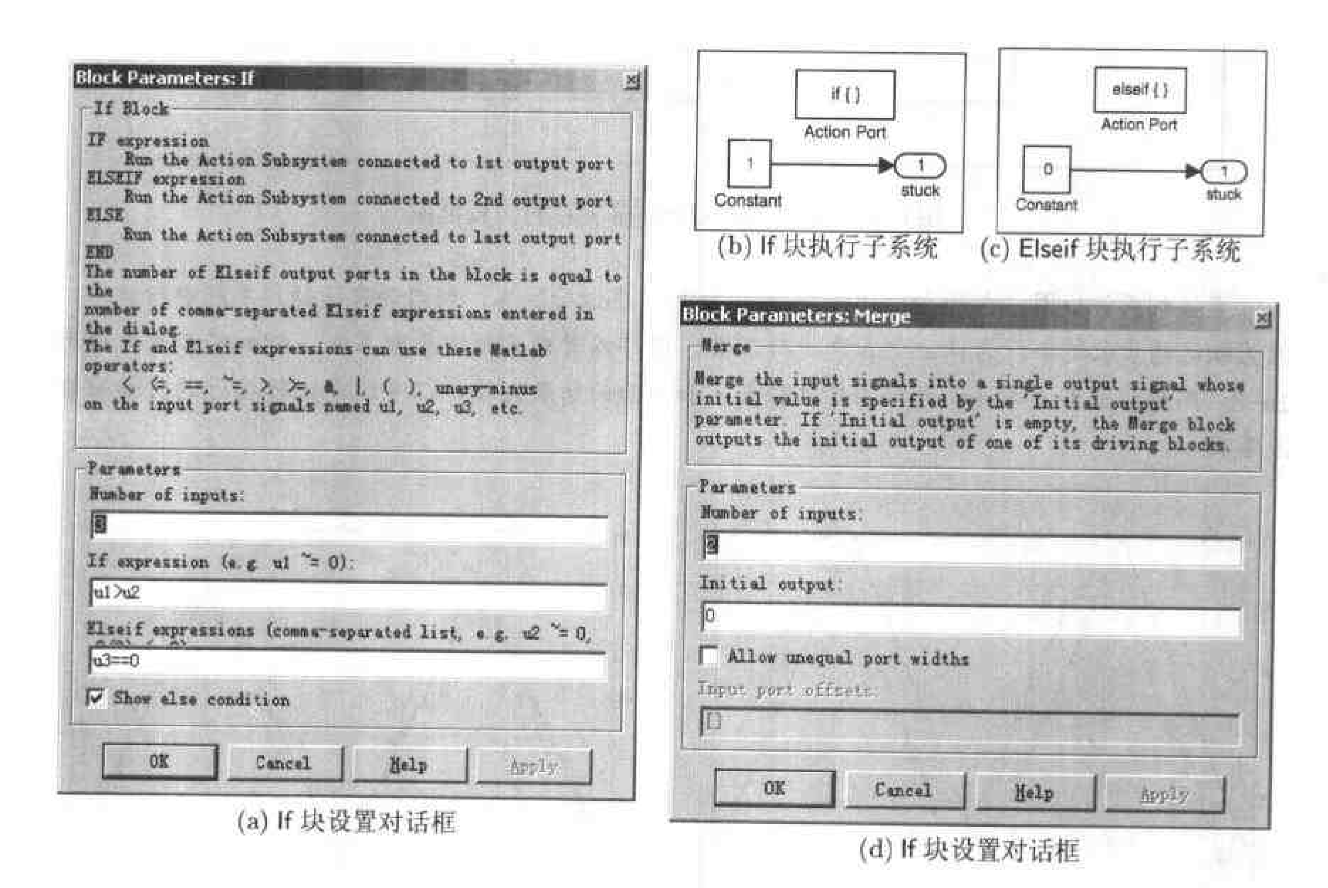


图 6-41 由 If 块构造的子系统



(a) If 块设置对话框

(b) If 块执行子系统

(c) Elseif 块执行子系统

(d) If 块设置对话框

图 6-42 输入信号参数设置对话框

接到 If 模块的输出端，如图 6-41 所示，这时它们的条件口分别自动变换成 If() 和 Elseif()，表示它们分别是这两个条件的执行系统。其实它们的内容很简单，分别如图 6-42 (b) 和 (c) 所示，其作用是将输出端口分别设置为 1 和 0。

可以从 Sinks 模块组中复制 Merge (合并信号) 模块，将两个 Action Port 的输入信号合并成一路，将其连接到本子系统的输出端子上。双击 Merge 模块，则将得出如图 6-42 (d) 所示的对话

框，可以将其初始输出 (Initial output) 设置为 0。

搭建了这样的子系统后，用它取代原来的 Stateflow 模块，则可以构造成如图 6-43 所示的新 Simulink 框图。

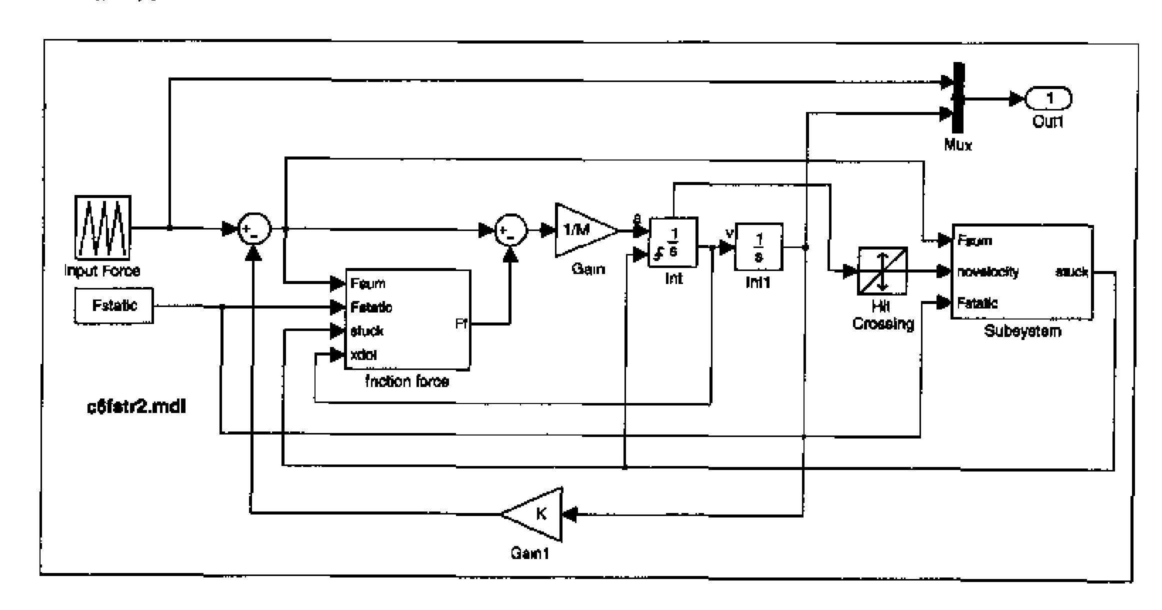
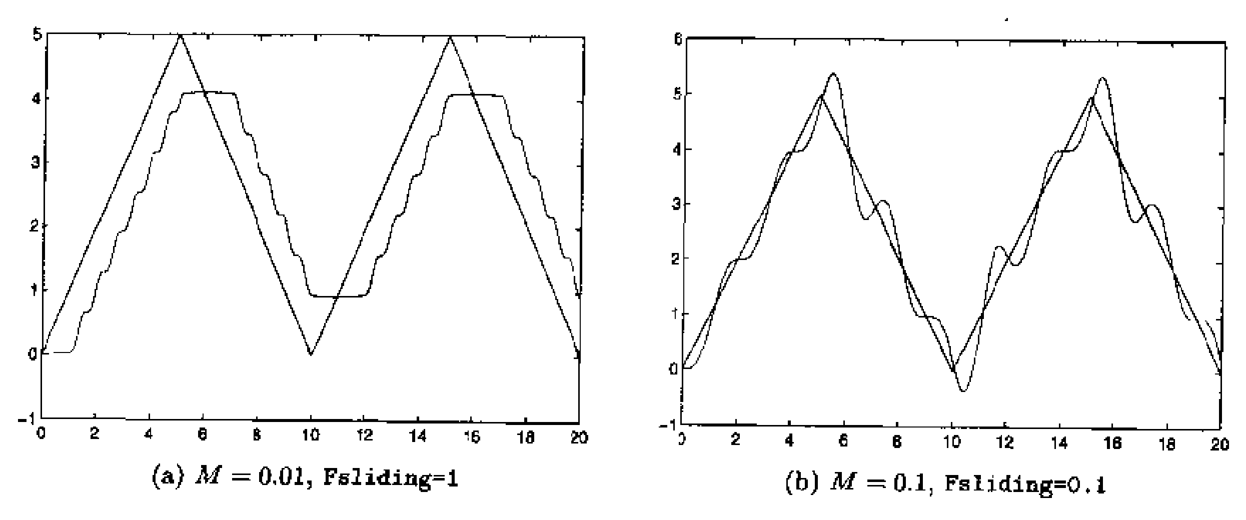


图 6-43 取代 Stateflow 模块后的仿真框图

直接利用该框图，在和 Stateflow 采用同样的算法进行仿真，则启动有问题，不能进行仿真，因为默认的步长过小。为保证仿真能进行下去，则可以考虑采用定步长算法。例如将步长选择为 1e-3，则得出如图 6-44 (a) 所示的仿真结果，和前面的结果比较，二者还是比较接近的。如果修改参数



(a) $M = 0.01$, `Fsliding=1` (b) $M = 0.1$, `Fsliding=0.1`

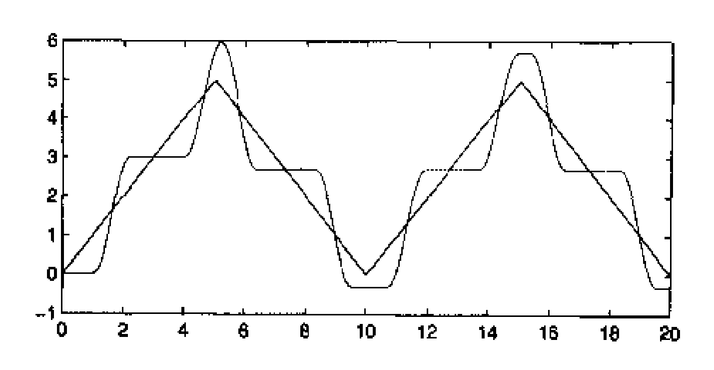
图 6-44 不同步长下的仿真曲线

```
>> M=0.1; Fsliding=0.1;
```

再进行仿真，则将得出如图 6-44 (b) 所示的曲线，和前面相比，不难发现这样仿真的结果有很大的差异。用带有 Stateflow 的模型重新进行仿真，再给出下面的语句，则可以同时绘制出该模型下

使用的仿真步长曲线和输入输出曲线，如图 6-45 所示。

```
>> subplot(211),plot(tout,yout)
   subplot(212),plot(tout(1:end-1),diff(tout))
```



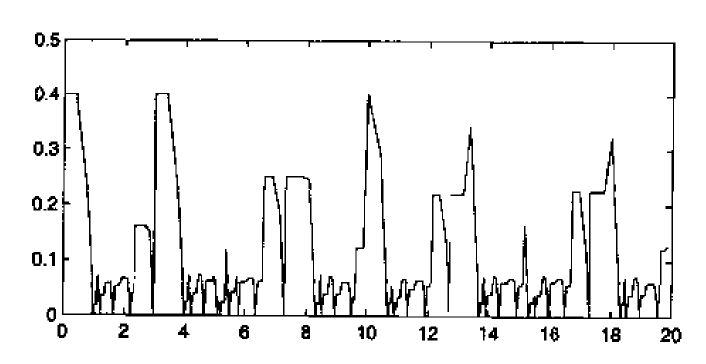


图 6-45 输入输出曲线与仿真步长

从二者关系可见，在输出曲线的转折点处计算步长必须足够小，经过观测应该达到 10^{-15} 级，如果采用定步长 10^{-3} 是不能满足要求的，故将得出错误结果。

〖例 6.20〗 前面介绍的转移模块还可以用下面的 S-函数来实现

```
function [sys,x0,str,ts]=c6fsfun(t,x,u,flag)
switch flag,
case 0
   [sys,x0,str,ts] = mdlInitializeSizes; % 初始化
case 3
   sys = mdlOutputs(u); % 输出量的计算
case { 1, 2, 4, 9 }
   sys = []; % 未使用的 flag 值
otherwise
   error(['Unhandled flag = ',num2str(flag)]); % 处理错误
end;
function [sys,x0,str,ts] = mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates = 0;  % 无连续状态
```

```
sizes.NumDiscStates = 0;  % 无离散状态
sizes.NumOutputs = 1;     % 输出个数为 1
sizes.NumInputs = 3;      % 输入个数为 3
sizes.DirFeedthrough = 1; % 输入不直接在输出中反映出来
sizes.NumSampleTimes = 1;
sys = simsizes(sizes);
x0 = []; % 设置初始状态为零状态
str = []; % 将 str 变量设置为空字符串
ts = [-1 0]; % 采样周期: [period, offset] 继承输入信号的采样周期
function sys = mdlOutputs(u)
if abs(u(1))>u(3)
   sys=1;
elseif u(2)==0
   sys=0;
end
```

这样就可以构造出如图 6-46 所示的 Simulink 框图，其作用与该转移模块是一致的，不幸的是，对这个摩擦系统来说，其效果和转移模块搭建的完全一致，有时不能正确对该系统进行仿真。

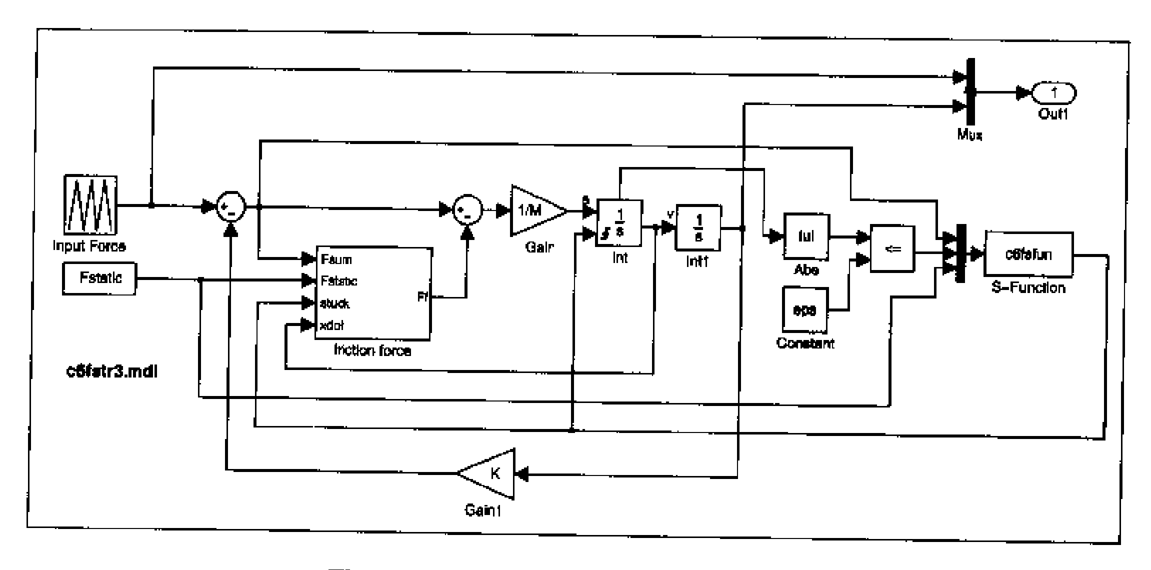


图 6-46 嵌入了 S-函数的摩擦系统模型

除了前面说明的转移结构，用 Simulink 的子系统还可以实现其他的流程，如循环结构等，使用方法和前面介绍的转移结构是类似的，所以在仿真中也可以充分利用这些流程结构。

从前面的例子可以看出，用 Simulink 自己提供的 If 模块能完成一些 Stateflow 善于表示的条件转移逻辑关系，但相对来说，这些模块只能表示简单的关系，且有时效果不算理想。其实 Stateflow 还能用于表示更复杂的逻辑关系，在图 6-47 中不加解释地给出了一个复杂的 Stateflow 框图[32]。当然熟悉和掌握 Stateflow 并非容易的事，文献 [32] 给出了

一些很好的例子，值得借鉴。

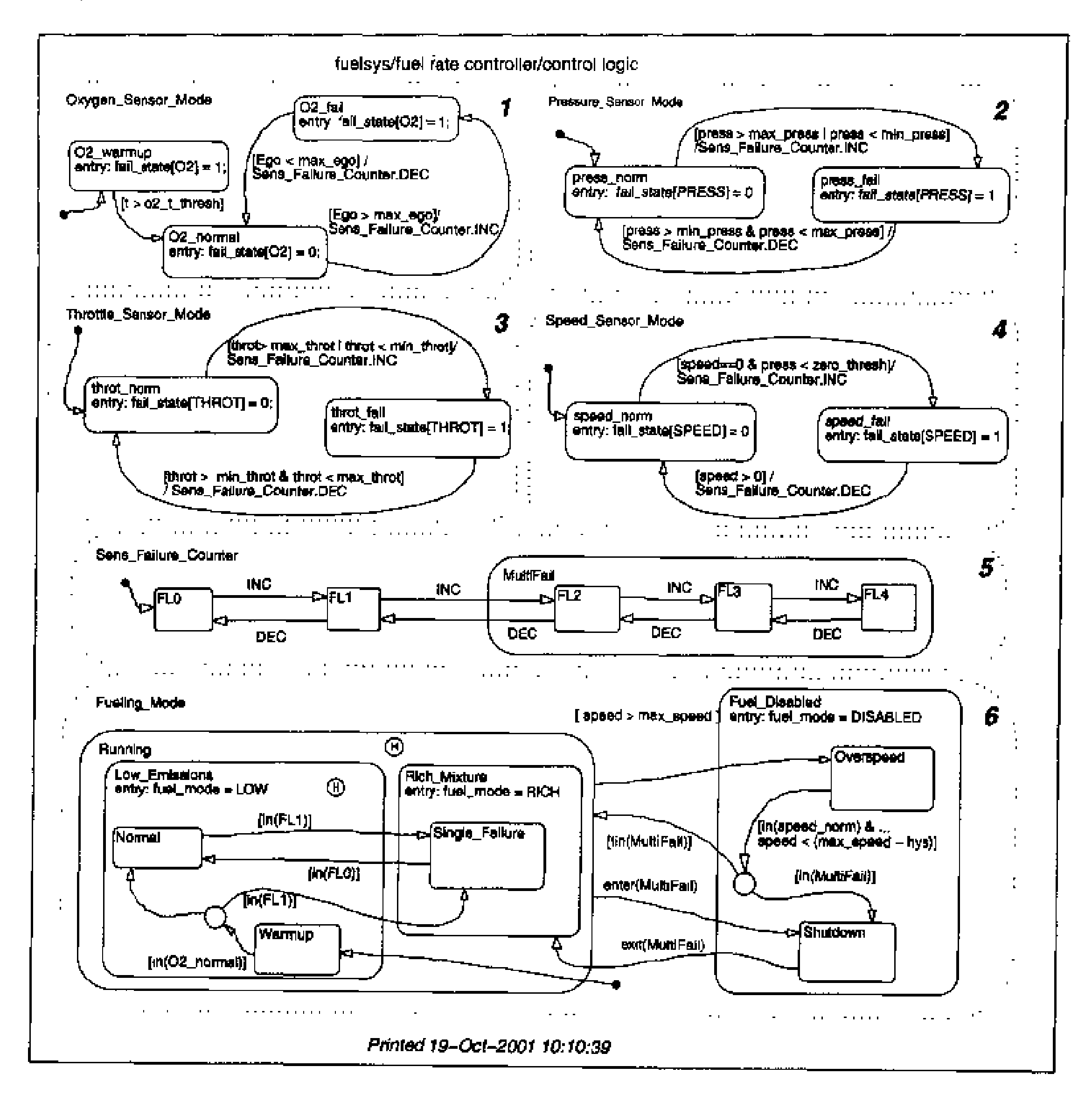


图 6-47 较复杂的 Stateflow 框图举例

6.5 MATLAB 虚拟现实工具箱及其应用

虚拟现实是一种可以创建和体验虚拟世界的计算机系统。虚拟世界是全体虚拟环境或给定仿真对象的全体。虚拟环境是由计算机生成的，通过视、听、触觉等作用于用户，使之产生身临其境的感觉或交互式视景仿真[45]。

虚拟现实必须是一个由计算机所产生的三维立体空间，用户可以和这个空间的对象进行交互，除观看外还可以操作其中部分对象，并可在空间中随用户的意志自由移动，进而产生相对的融入感和参与感[56]。Burdea 公司提出了广为认可的 3I 定义，亦即：

- **Immersion** (沉浸度)：人们全心投入一件事情时，会达到浑然忘我的境界，完全无

视外界的环境。虚拟现实就是借助这种心理让人摆脱现实环境的压力，进入计算机模拟的虚拟现实。

- **Interactive (交互性)**：真实世界中，人可以和周围的环境交互，所以虚拟现实就是要把这种人与环境间的交互性加入虚拟现实中，让虚拟现实更为真实。
- **Imagination (想像力)**：虚拟现实就是借助人类的想像力，将虚拟现实和真实的实物联想在一起。

新出现的基于 MATLAB 的虚拟现实工具箱允许 MATLAB/Simulink 使用虚拟现实的图形技术，使得用户直接将仿真结果以虚拟现实的形式显示出来。本节将首先介绍虚拟现实的基本概念，再介绍虚拟现实模型语言 VRML 程序的生成方法，然后介绍虚拟现实工具箱的基本功能，包括在 MATLAB 环境下和 Simulink 下仿真结果的虚拟现实显示方法及应用。

6.5.1 虚拟现实工具箱的安装与设置

虚拟现实工具箱是随着 MATLAB 6.1 版本推出而正式推出的，以前虽然有早期版本的存在，但一直未有广泛的流传。

虚拟现实工具箱的安装有自己的特色，应该分为下面几个步骤：

- **虚拟现实工具箱安装** 工具箱本身可以和 MATLAB 一起安装，如果在允许的安装组件列表中有 Virtual Reality Toolbox 选项，则可以直接安装。如果以前没有安装此工具箱，则可以重新启动 MATLAB 的安装程序，输入允许安装该工具箱的 PLP，再选择虚拟现实工具箱安装选项，就可以安装该工具箱。
- VRML **预览器安装** VRML 语言构造虚拟现实场景可以通过基于 Web 的预览器来显示，可以选择虚拟现实工具箱中提供的 blaxxun Contact 的 VRML 插件，将其通过下面的命令将 VRML 预览器建立起来。

```
>> vrinstall -install viewer
Do you want to use OpenGL or Direct3d acceleration? (o/d)
```

  从上述的提示中可以选择其一，例如选择 o，表示 OpenGL 加速方式，则将得出如图 6-48 所示的对话框，该对话框将引导用户进行全部的预览器安装全部预览器程序，安装完成后将给出下面的提示：

```
Starting viewer installation...
Done.
```

- VRML **程序编辑器安装** 作为虚拟现实工具箱的一个组成部分，提供了 V-Realm Builder 2.0 版作为 VRML 程序编辑器，其安装方法是在 MATLAB 命令窗口中给出如下命令：

```
>> vrinstall -install editor
```

- **安装检验** 安装完插件和编辑器后，可以运行下面命令来检查安装是否完成：

```
>> vrinstall -check
    VRML viewer:    installed
    VRML editor:    installed
```

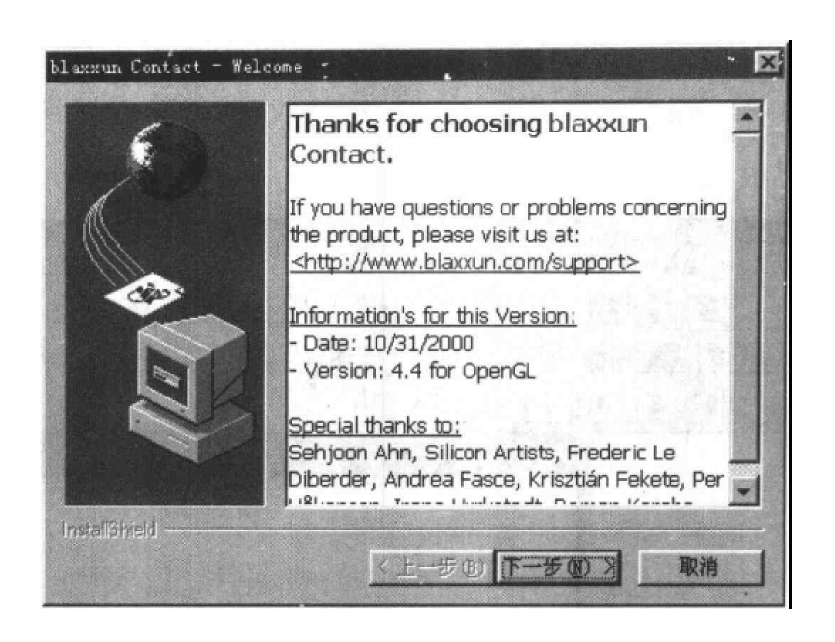
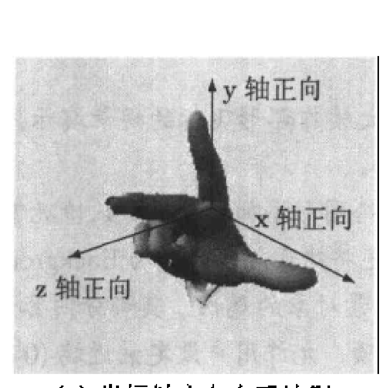


图 6-48 VRML 预览程序安装界面

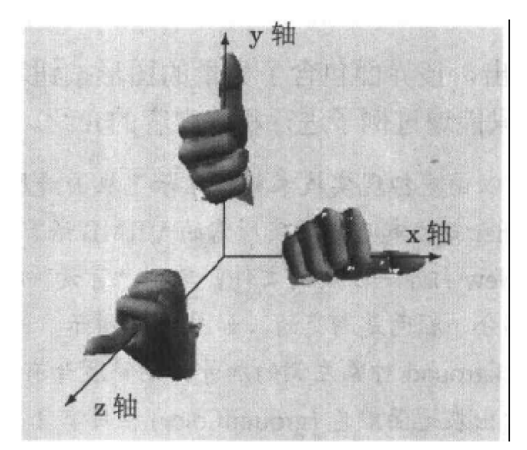
6.5.2 VRML 语言程序设计入门与举例

VRML (Virtual Reality Modelling Language) 是一种常用的虚拟现实描述语言，在 MATLAB 的虚拟现实工具箱中主要采用这样的语言来描述虚拟现实。

VRML 语言下描述三维空间时，其坐标框架满足右手法则，如图 6-49 (a) 所示，即右手的拇指、食指、中指相互垂直，则它们的指向分别构成了 x, y, z 轴。从该图可见，其坐标轴排列顺序和常规使用的坐标系不完全一致，所以应该注意。



(a) 坐标轴方向右手法则



(b) 坐标轴旋转方向

图 6-49 右手法则示意图

定义了坐标轴方向以后，右手拇指指向坐标轴的正方向，其余四指握拳后所指的方向就是沿该坐标轴旋转的方向，其示意图如图 6-49 (b) 所示。

V-realm Builder 2.0 版是一个编辑 VRML 程序的实用可视化工具，它允许用户搭建虚拟现实所需要的场景和素材，该软件在新发布的 MATLAB 6.1 的虚拟现实工

具箱中也包含免费的版本，安装虚拟现实工具箱时会自动地将其安装到工具箱下的 vrealm\program 目录，直接启动其中的 vrbuild2.exe 文件即可，这时将得出如图 6-50 所示的界面。

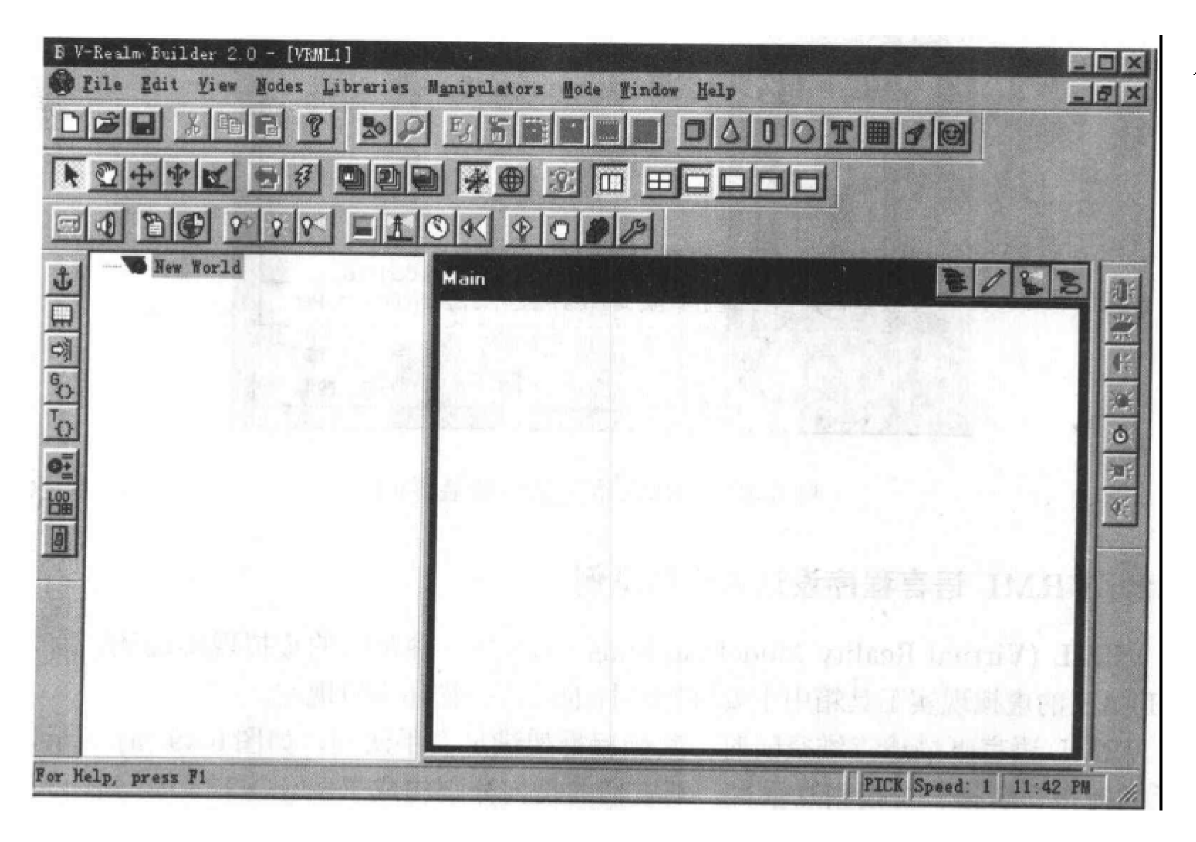


图 6-50　V-realm Buide 2.0 运行界面

可以看出，该界面包含了大量的图形按钮和菜单项，所以这里不可能详细介绍该程序的使用，只能通过例子进行相关内容的介绍。

【例 6.21】 可以由虚拟现实技术模拟显示飞机围绕摩天大楼作环形飞行的视景演示。首先应该用 V-realm Builder 建立起虚拟现实所需的 VRML 语言文件。

用 File| New 打开一个新的文件，单击“背景”按钮给图形添加背景，默认情况下，背景分为天和地两个部分，颜色是渐变的，如图 6-51 所示，同时还将建立起一个名为 Background 的对象。

单击 Background 对象左侧的加号，则将展开有关背景对象的属性，其部分内容如图 6-52 (a) 所示，可以看出在地面颜色 (groundColor) 栏目有 4 个子项，允许用户设定最近端 (0 级) 到最远端 (3 级) 的颜色值，例如双击 [3] 栏目，将得出如图 6-52 (b) 所示的对话框，用户可以从中选择相应的颜色。还可以通过类似的方法设置其他的颜色。

从 V-realm Builder 2.0 的界面的第一行工具栏可见，该软件提供了大量的对象添加功能，例如在窗口中添加圆柱体、长方体、圆锥等；除此之外，V-realm Builder 还提供了大量的现成对象，例如可以选择 Library | Import from | Objects Library 菜单项 (如图6-53 (a))，则得出各种各样的对象库，如图 6-53 (b) 所示。

可以从建筑 (Archetecture (Building)) 组中选择摩天大楼 (Skyscraper) 对象，将之拖动到虚拟

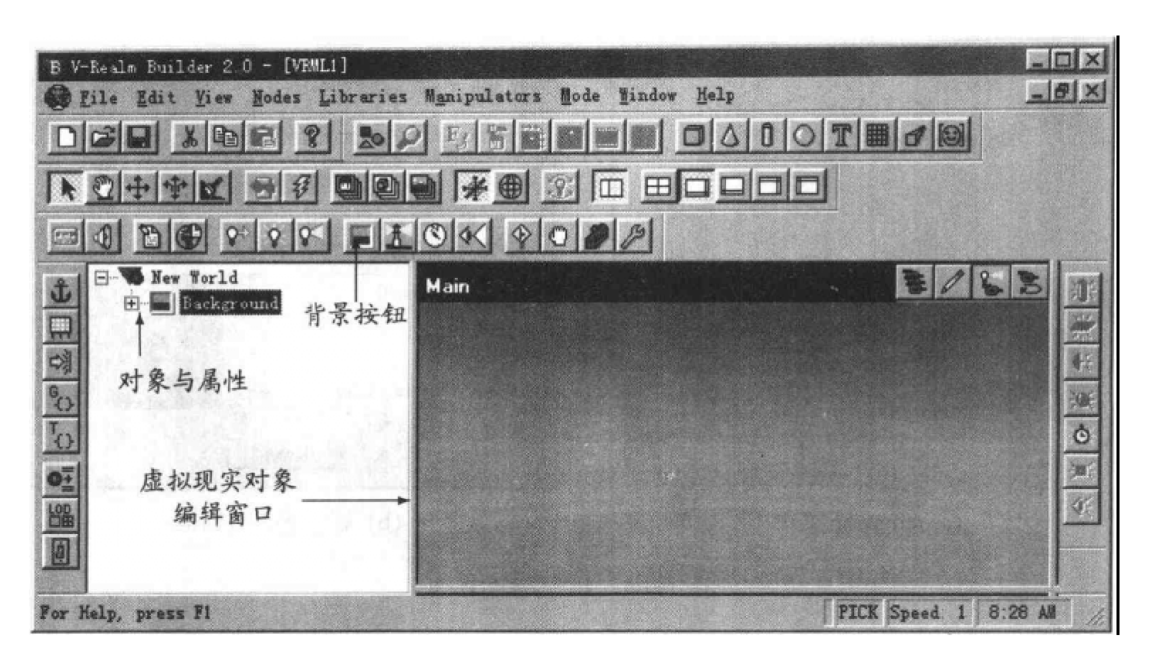
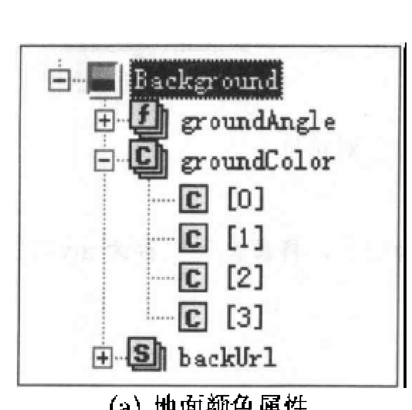
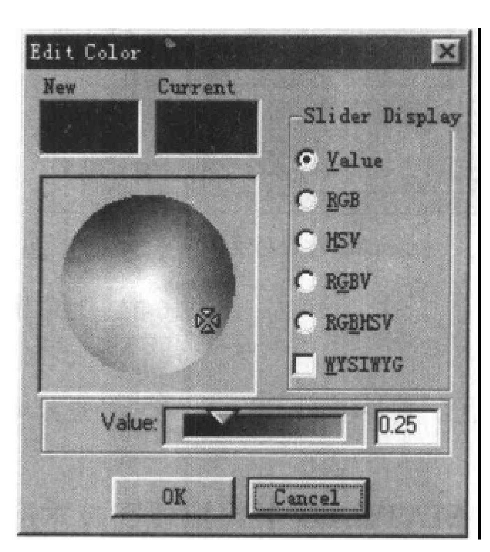


图 6-51 V-realm Buide 2.0 添加背景的结果



(a) 地面颜色属性

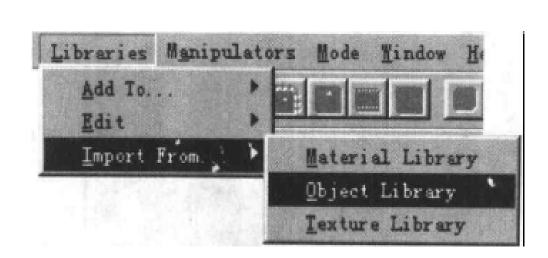


(b) 颜色设置对话框

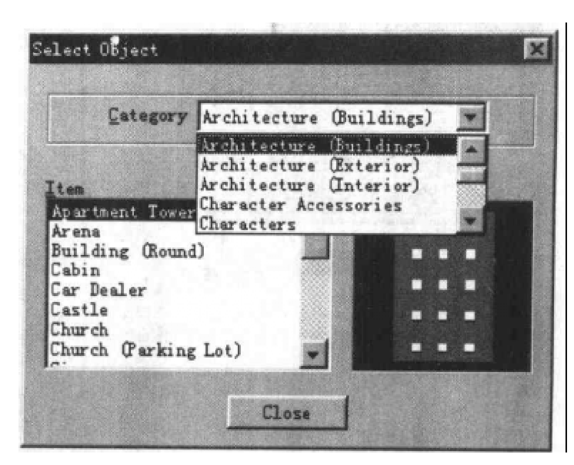
图 6-52 属性颜色设置

现实对象编辑窗口中，类似地还可以选择飞行交通工具 (Transportation (Air)) 组中 Boeing 737 对象，就可以在总的框架中加入两个对象，都标为 Transform，分别改写其名称为 Skyscraper 和 Plane。事实上，直接建立起来的这两个对象尺寸过大，显示起来有些问题，如果双击 Skyscraper 标识下的 scale 栏目，则可以得出如图 6-54 (a) 所示的对话框，可以减小各轴的标度，使得整个图形可读。另外还可以修改对象所在中心的选项，亦即双击该对象的 center 栏目，则得出如图 6-54 (b) 所示的对话框。

设置 Skyscraper 对象的 center 设置为 4 个参数 (0, 0, -9, 0.1)，将其 scale 参数设置为 (0.1, 0.28, 0.1, 0.1)，还可以将 Plane 对象的 center 属性设置为 (-4.7, -0.6, 1, 0.1)，且其 scale 属性为 (0.15,

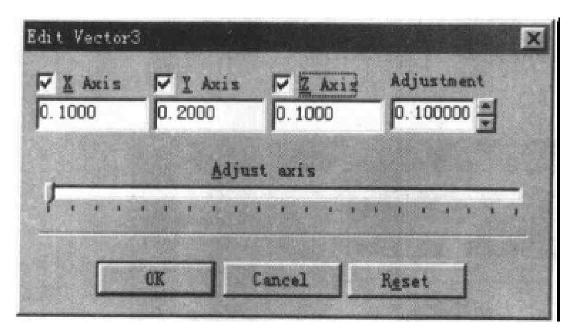


(a) Libraray 菜单

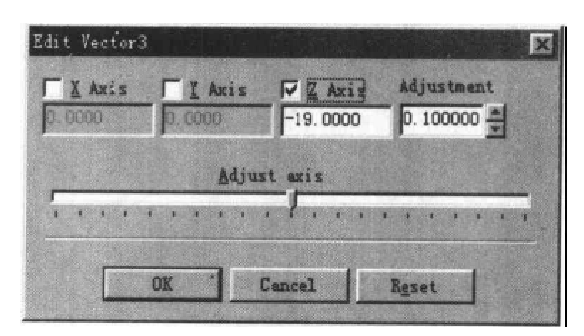


(b) 对象选择对话框

图 6-53 属性颜色设置



(a) scale 设置对话框



(b) center 设置对话框

图 6-54 对象属性设置对话框

0.15, 0.15, 0.1)，则将得出如图 6-55 所示的虚拟现实素材了，将该素材保存为 myvr1.wrl 文件。

6.5.3 在 MATLAB 下虚拟现实技术应用

前面介绍的 myvr1.wrl 文件是由 V-realm Builder 软件直接绘制出来的静态图形，用户可以用 VRML 语言编写程序，使其“动”起来，但这要求用户掌握该语言的编程方法和技巧，对一般用户不是件简单的事。

MATLAB 提供的虚拟现实工具箱中提供了一系列简单函数，直接对 *.wrl 文件中描述的对象的属性直接操作，其方便程度类似于 MATLAB 对自己对象操作一样简单。在 MATLAB 下调入并获取整个虚拟现实文件 (称为“世界”) 和各个对象 (称为“节点”) 的属性可以采用下面几个语句来实现：

(1) 打开虚拟现实的 *.wrl 文件用 vrworld() 函数来实现，例如用下面的函数就可以将虚拟现实文件 myvr1.wrl 中描述的世界的句柄赋给 myworld 变量。

```
>> myworld=vrworld('myvr1.wrl');
```

(2) 导入虚拟现实世界可以使用 open() 命令实现，在 Web 浏览器中显示虚拟现实世

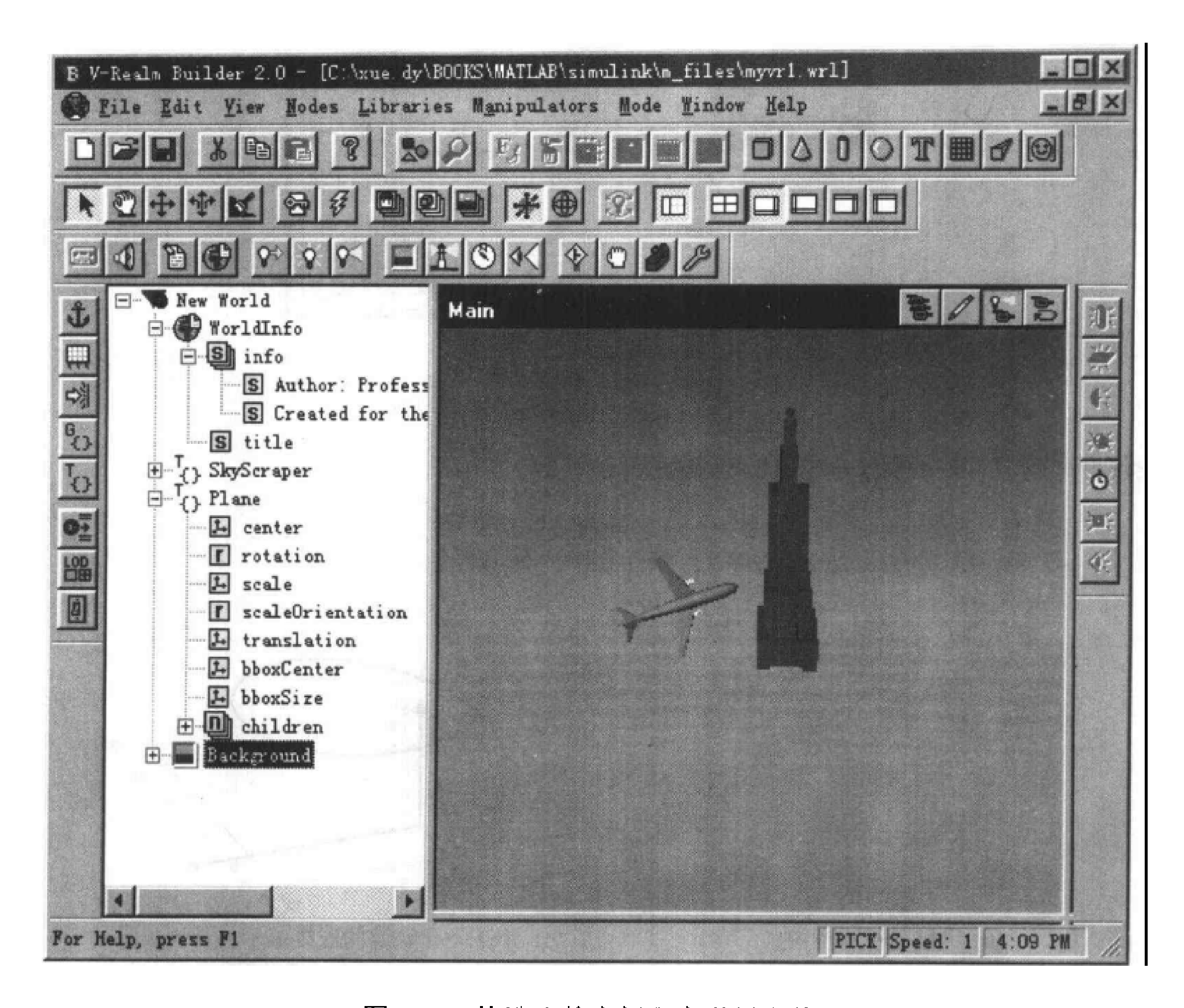


图 6-55 构造出的虚拟现实世界文件

界可以使用 view() 函数实现，它将自动地打开一个网页浏览器，如 Internet Explorer 浏览器。

```
>> open(myworld) % 用世界的句柄导入该世界
   view(myworld) % 打开浏览器界面，显示该世界
```

虚拟现实的显示是基于网页浏览器的，所以如果计算机未联网，则运行 view() 函数时将出现如图 6-56 所示的提示，单击“重试”按钮就可以建立起初始网页，直接显示虚拟现实结果。

在即将正式推出的 MATLAB Release 13 (虚拟现实工具箱3.0 版) 中，虚拟现实工具箱提供了独立的虚拟现实显示工具，不再依赖 Internet Explorer 浏览器，所以显示起虚拟世界效果更好。

(3) 导入了虚拟现实世界后，就可以用 vrnode() 函数获得各个节点的句柄，并采用 set() 函数修改各个节点属性，以便进行虚拟现实显示。下面将通过例子来演示虚拟现实工具箱的使用方法。

【例 6.22】 利用前面建立起的“世界”文件，假定飞机从初始位置，以摩天大楼的中心为圆心，按上升的螺旋函数围绕大楼飞行，如图 6-57 (a) 所示，则可以用下面的思路来编写程序，将仿真

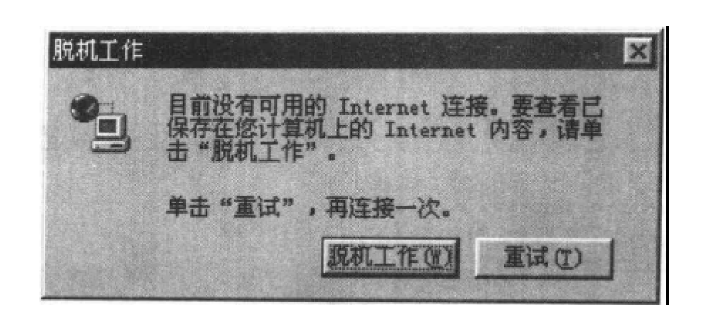
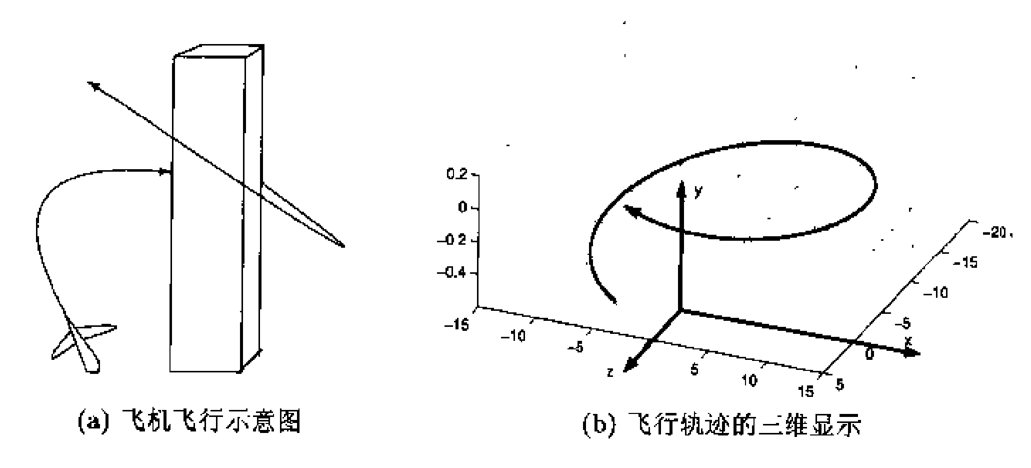


图 6-56 网页错误信息提示

结果按虚拟现实的方式显示出来。



(a) 飞机飞行示意图　　(b) 飞行轨迹的三维显示

图 6-57 仿真目的示意图和轨迹三维显示

已知起始点为 (-4.7, -0.6, 1)，圆心位置为 (0, 0, -9)，并假设飞机的运动轨迹为

$$x = 11\cos(t + 118°),\ y = -0.6 + 0.1t,\ z = -9 + 11\sin(t + 118°)$$

其中参数变量 $t \in (0, 360°)$。注意，在 MATLAB 下计算正弦数据需要使用弧度单位，所以应该进行变换。用下面的命令绘制出轨迹的三维图，如图 6-57 (b) 所示。

```
>> t=0:.1:2*pi; t0=118*pi/180; % 设置 t 向量，将角度变弧度
   x=11*cos(t+t0); y=-0.6+0.1*t; z=-9+11*sin(t+t0);
   plot3(x,z,y), grid, set(gca,'box','off')
   set(gca,'xdir','reverse','ydir',reverse') % 常规坐标的 x,y 轴反向
   view(-67.5,52) % 旋转坐标系到一个更好的视角
```

使用下面的 MATLAB 命令，就可以导入前面建立的 myvr1.wrl 文件，显示出所建立起来的虚拟世界了。

```
>> myworld=vrworld('myvr1.wrl');
   open(myworld); view(myworld);
   plane=vrnode(myworld,'Plane'); % 获得飞机对象句柄
   set(plane)   % 显示飞机对象的属性
```

```
scaleOrientation      4 doubles
removeChildren        (not settable)
children              (not settable)
center                3 doubles
translation           3 doubles
addChildren           (not settable)
bboxSize              3 doubles
bboxCenter            3 doubles
rotation              4 doubles
scale                 3 doubles
```

可以看出，使用 set() 函数可以显示出所有可以修改的属性名称和属性值的形式。修改这些属性与修改普通 MATLAB 句柄对象的属性没什么不同。比如这里较感兴趣的属性有两个：一个是其 center 属性，一个是其 rotation 属性，前者设置其位置的值，后者可以用来表示飞机的飞行方向，它可以是轨迹的切线方向。

运行其中的 view() 函数则将自动打开一个浏览器界面，如图 6-58 所示。该窗口将 myvrl.wrl 文件中描述的静态图形显示出来。该界面下部有一个控制工具，用户可以用它调整视角，从不同的角度观察虚拟现实效果的显示。

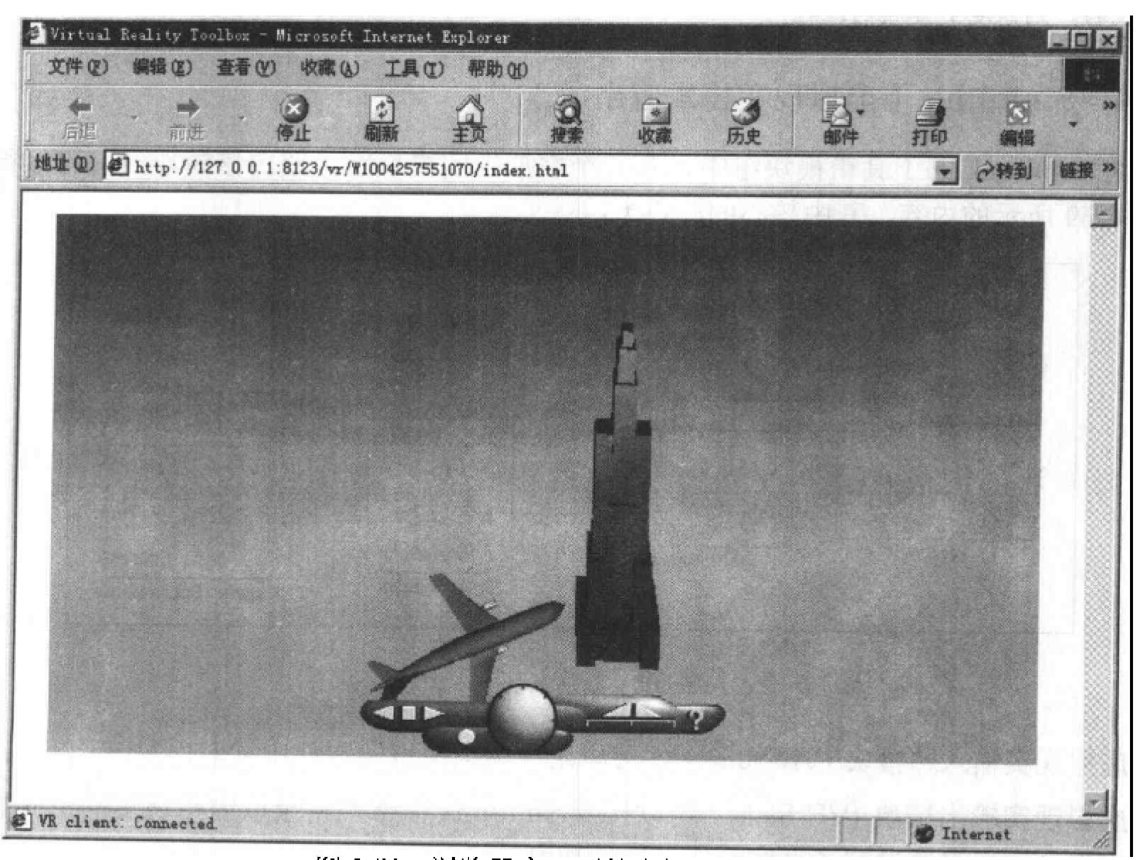


图 6-58 浏览器窗口下的虚拟现实显示

在下面的程序中只考虑飞机的位置，取一系列坐标点 (x_i, y_i, z_i)，然后执行循环，在循环中每

一步都更新飞机的坐标位置点 (即设置其 center 属性), 然后给出 0.05 秒的延迟。

```
>> for i=1:length(x)
      set(plane,'center',[x(i),y(i),z(i)]); pause(0.05);
   end
```

程序执行后感觉飞机真的绕大楼飞行。然而，这样的飞行演示还是有缺陷的，即飞机的本身方向不发生任何变化，只进行平移运动，因为定义的只是其中心位置。如果想同时让飞机头部指向飞行的前方，可以进行角度设置。为简单起见，忽略在 y 轴 (高度) 上的位移，则飞机在 x-z 平面上作顺时针圆周运动。取 x 和 z 轴变化率商的反正切就能正确设置飞机方向，这时相当于飞机在虚拟现实坐标系内绕 y 轴作正方向的旋转，可以给出如下命令:

```
>> v(1:3)=[0;1;0]; % 标明按 y 轴进行旋转
   for i=1:length(x)-1
      a1=x(i+1)-x(i); c1=z(i+1)-z(i);
      v(4)=atan2(a1,c1);% 设置旋转角度
      set(plane,'rotation',v,'center',[x(i),y(i),z(i)]),
      pause(0.05); % 同时设定位置和角度
   end
```

这时的显示效果已经大大改进，但这样的设置还是近似的，因为忽略了 y 轴上的位移。如果再考虑该位移，则需要更复杂的编程。

6.5.4　Simulink 下虚拟现实技术应用

在 Simulink 的工具箱模块组中，有一个虚拟现实模块组，双击该模块组，可以得出如图 6-59 所示的内容，其中核心的模块为：

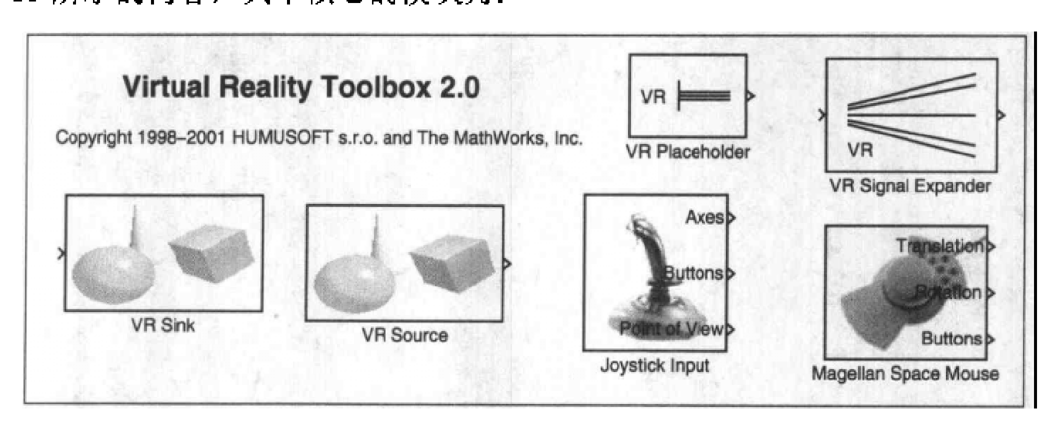


图 6-59　虚拟现实模块组

- **虚拟现实输入源模块** (VR Source)　可以用于由虚拟现实文件中读取信息。
- **虚拟现实输出模块** (VR Sink)　可以接受 Simulink 信号，结果以虚拟现实的形式显示出来。这里将通过例子来演示该模块的使用。

【例 6.23】 仍然考虑前面飞机飞行的例子，已经建立起了 myvr1.wrl 文件，来描述所需的虚拟世界。打开一个空白的模型窗口，将 VR Sinks 模块复制到该窗口中，双击该模块，将得出如图 6-60

所示的对话框，在 associated file (关联文件) 中填写文件名 myvrl.wrl, 则在 VRML Tree (VRML

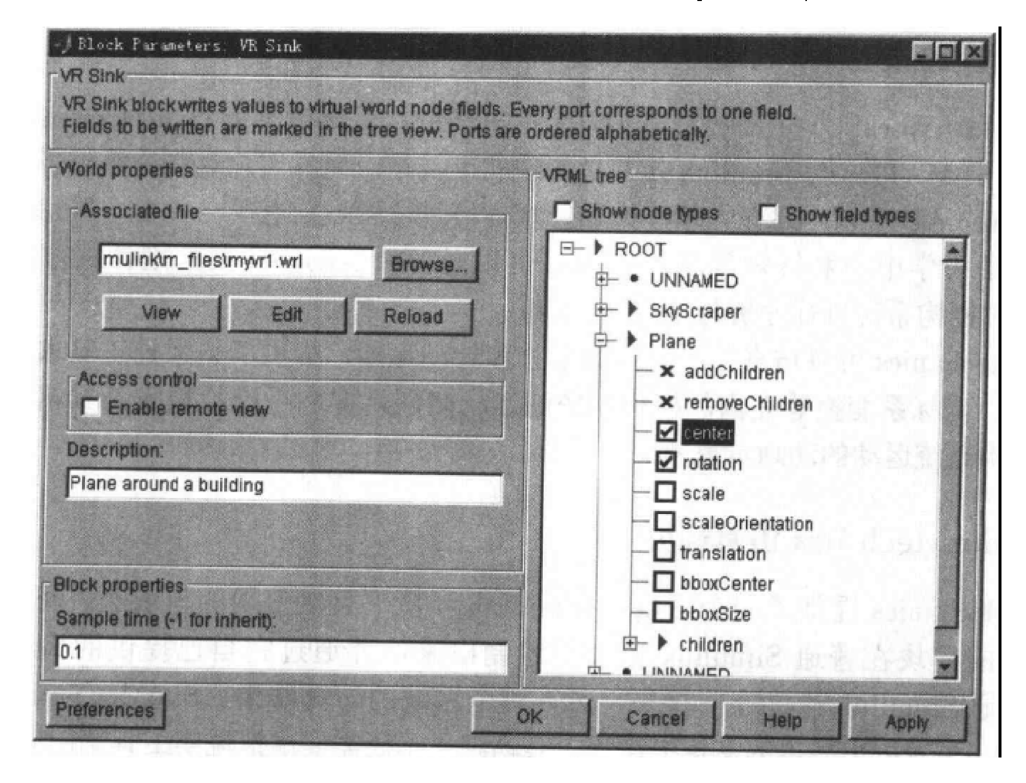


图 6-60 虚拟现实输入模块对话框

语言节点树) 列表框中将显示出全部节点的列表，单击 Plane 节点左侧的 + 号，则将把该节点的属性全部列出。和前面一样，因为想修改其 center 和 rotation 两个属性，所以在该对话框中应该选中这两个属性，在单击 OK 按钮，则可以给该模块设置两个输入端子，`Plane.center` 和 `Plane.rotation`。

单击 View 按钮就可以打开网页浏览器，导入虚拟现实的图形显示，如图 6-58 所示，其作用类似于前面介绍的 `view()` 函数.

这样就可以依照前面的要求建立起整个仿真过程的 Simulink 模型如图 6-61 所示。启动仿真过程将得到和前面例子同样的效果。从这个例子中可以发现，似乎这样处理比 MATLAB 下的调用方式更简洁。

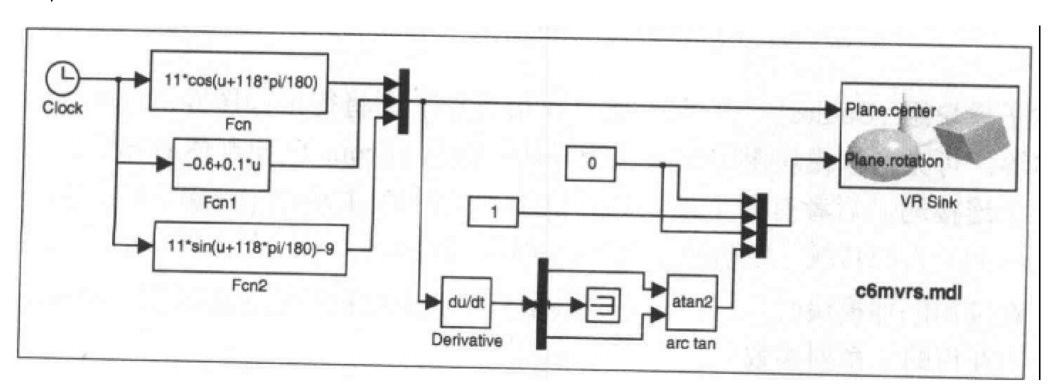


图 6-61 飞机飞行仿真模块框图

6.6 SimMechanics —— 机构系统模块集及应用

6.6.1 物理建模与机构系统仿真

The MathWorks 公司于 2001 年 10 月推出了机构系统模块集 (SimMechanics Blockset)，借助于 MATLAB/Simulink 及其虚拟现实工具箱，允许用户对机构系统进行仿真，这标明 MATLAB 系列产品在物理建模 (或概念性建模) 领域前进了一大步。SimMechanics 利用牛顿动力学中力和转矩等基本概念，可以对各种运动副连接的刚体进行建模与仿真，实现对机构系统进行分析与设计的目的。

SimMechanics 可以仿真三维系统的平移和转动运动，提供了一系列工具求解带有静力学约束、坐标系变换等在内的机构系统的运动问题，并利用虚拟现实工具箱提供的功能显示机构系统运动的动画示意图。

6.6.2 SimMechnics 仿真简介

SimMechanics 提供了一个可以在 Simulink 环境下直接使用的模块集，可以将表示各种机构的模块在普通 Simulink 窗口中绘制出来，并通过它自己提供的检测与驱动模块和普通 Simulink 模块连接起来，获得整个系统的仿真结果。SimMechanics 必须在 MATLAB 6.1 及以上版本的支持下运行，其动画显示还需要虚拟现实工具箱的支持。

作为 Simulink 下的一个应用程序，可以从 Simulink 浏览器中直接打开 SimMechanics 模块组，也可以在 MATLAB 命令窗口中由 `mechlib` 命令打开该模块组，后者将得出如图 6-62 所示的模块组。可见，该模块组中包含下面几个子模块组：

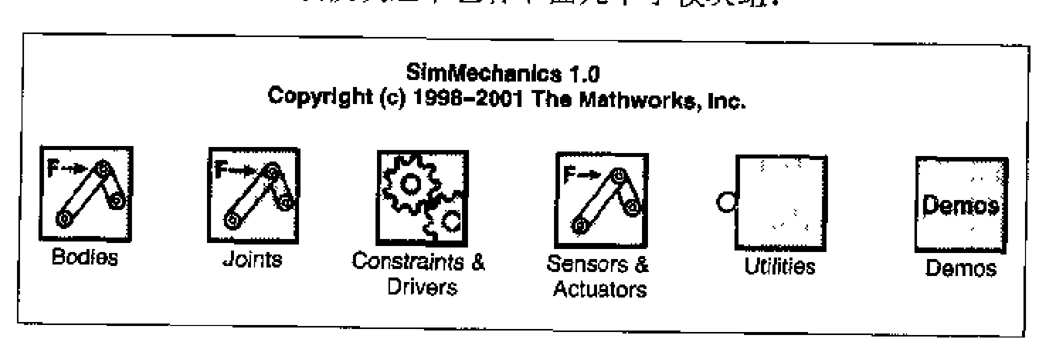
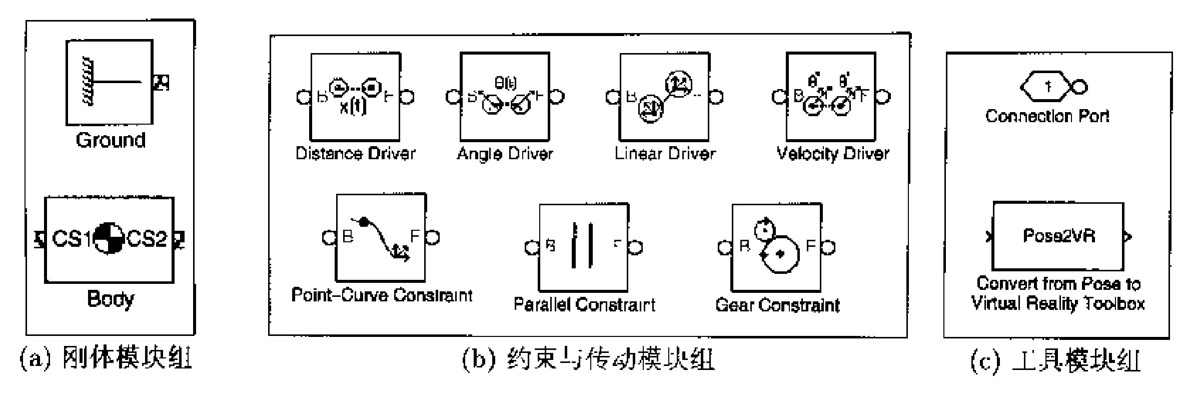


图 6-62 SimMechnics 模块组

- **刚体子模块组** (Bodies) —— 双击该模块组图标，则将得出如图 6-63 (a) 所示的模块组内容。可见，该模块组中共有两个模块：机架 (Ground) 和刚体 (Body)，其中前者有一个连接端，后者有两个连接端，其中 B 端称为主动端 (base)，F 端称为从动端 (follower)。按照传统系统仿真的概念，也可以将 B 端理解为输入端，F 端为输出端。

  在使用刚体模块时，用户还应该根据实际情况设置包括刚体质量、位置、方向和坐标系在内的一系列参数。

- **约束与驱动模块组** (Constraints & Drivers) —— 双击该图标则得出如图 6-63 (b) 所示的模块组内容。在该模块组中有静力学约束的模块，如齿轮约束 (Gear Constraint)、



(a) 刚体模块组　(b) 约束与传动模块组　(c) 工具模块组

图 6-63　常用子模块组

平行约束 (Parallel Constraint) 和曲线约束 (Point-Curve Constraint)，另外，本模块集中还包含各种传动模块。

- **辅助工具模块组** (Utilities) —— 该模块组的内容如图 6-63 (c) 所示，这里的模块允许在其他模块中添加节点，或将信息转换成虚拟现实工具箱用的数据。
- **运动副模块组** (Joints) —— 该模块组的内容如图 6-64 所示，提供了各种运动副的图标，可以用这些运动副来连接刚体，构造所需的机构。在该子模块组中提供了单自由度的转动副 (Revolute)、单自由度移动副 (Prismatic)、球面副 (Spherical，有三个自由度)、平面副 (Planar)、万向轴节 (Universal)、圆柱副 (Cylinder)、螺旋副 (Screw) 及六自由度 (Six-DoF) 等。

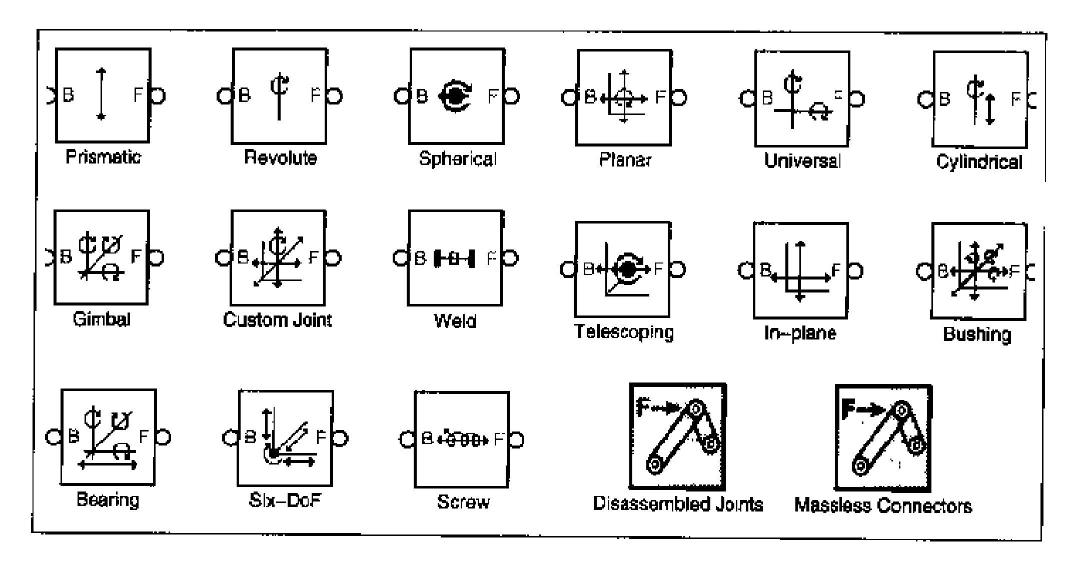


图 6-64　运动副模块组

- **检测与驱动模块组** (Sensors & Actuators) —— 该模块组的内容如图 6-65 所示，该模块组中的模块用来和普通的 Simulink 模块交互信息，例如可以将刚体检测模块 (Body Sensor) 连接到刚体的附加输出端，用以检测刚体的线速度、角速度、位置和加速度等信息，将其输出端连接到示波器上显示出来。驱动模块用来给机构添加 Simulink

输入量，例如，用户可以用普通的 Simulink 模块搭建一个力信号，然后通过刚体驱动模块 (Body Actuator) 将该力信号施加到相应的刚体上。这个过程看起来较复杂，事实上，要想使得 SimMechanics 模块和普通 Simulink 模块进行数据交换，就必须使用这样的中间环节。

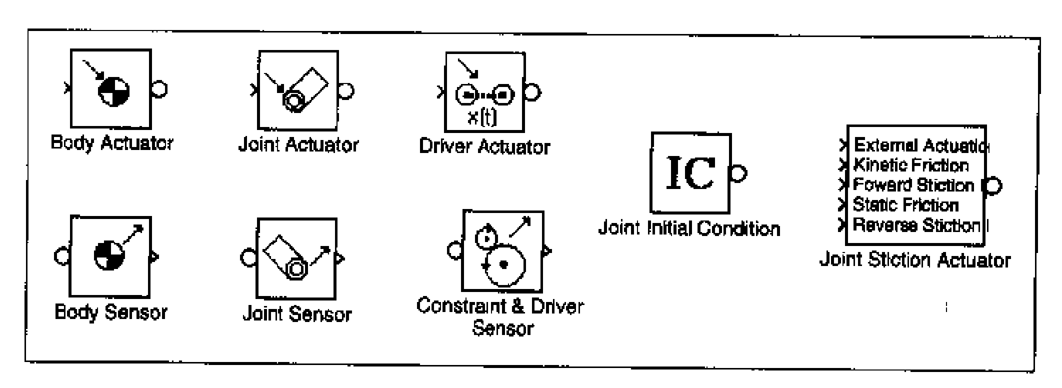


图 6-65 检测与驱动模块组

6.6.3 机构系统仿真举例

机构系统仿真是很重要的，例如，机器人系统的机械部分经常可以简化成机构系统的仿真[6]，而 SimMechanics 和 MATLAB/Simulink 之间的有机结合又允许将机构系统和电控部分在同一仿真框架下同时进行，这样就使得整个系统仿真的工作更容易、有效。

本节将以平面四连杆机构的建模与仿真为例，介绍 SimMechanics 在机构系统为例建模与仿真中的应用，通过这个例子，相信读者会对解决类似的问题有一定的认识，可以更好地利用这一工具求解其他的机械系统建模与仿真的问题。

【例 6.24】 考虑如图 6-66 所示的平面四连杆机构的运动简图[46]，整个连杆机构的几何尺寸在图中

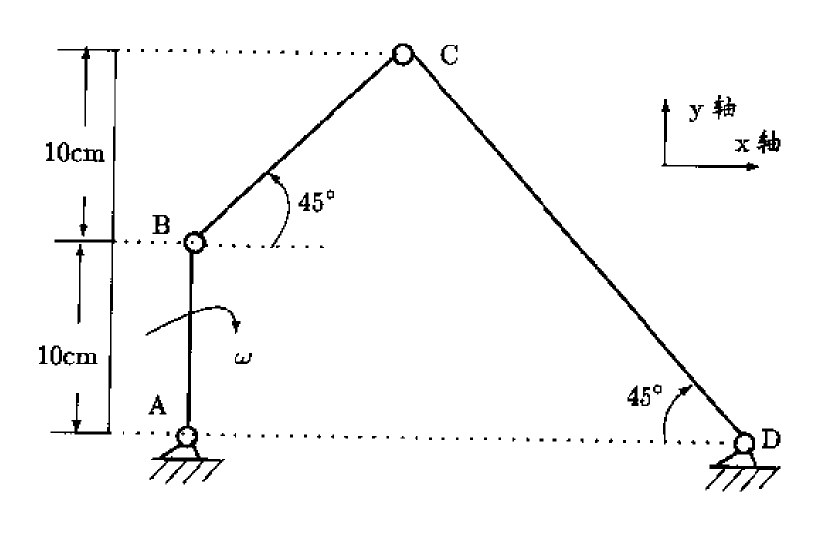


图 6-66 四连杆机构运动简图

给出。从该机构运动简图可见，整个系统有两个固定机架，三个刚体的连杆通过四个单自由度转

动副和这两个固定机架相连，假设连杆 AB 沿 A 轴以 ω 的角速度旋转，可以取其为正弦信号，试分析 C 点的运动轨迹。

对这样的机构系统，我们将分以下几个步骤来求解：

(1) **仿真框图绘制**

在该机构中，虚线表示的 AD 线可以认为是一个固定的杆，故这样的机构称为四连杆机构。利用 SimMechanics 中提供的模块，不难建立起该系统的 Simulink 框图的雏形，如图 6-67 所示。

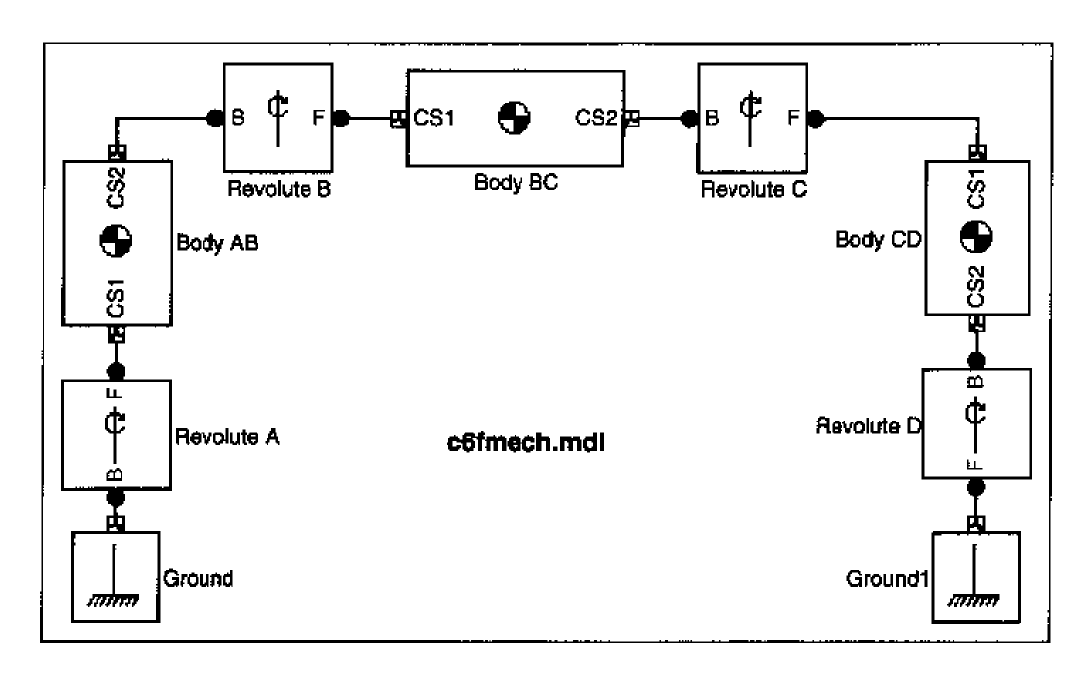


图 6-67 平面四连杆机构的 SimMechnics 表示

在该框图中，首先需要绘制出固定机架，可以采用刚体模块组中的 Ground 模块来表示，然后从 Joints 模块组中复制 Revolute 模块，构造出第一个转动副，再从 Body 模块组中复制 Body 模块，依此类推，就可以将所需的模块都复制到此模型窗口中。复制完模块后，用类似于普通 Simulink 模块连接的方法，就可以将这些模块连接起来。在建立该框图时，我们只是将相应的模块按照示意图中指定的方式堆砌起来，并未设置各个模块的参数及与 MATLAB/Simulink 进行信息交换的方式，另外，根据原来的要求，还需要给 AB 杆加一个外加的角速度，所以这样直接建立起来的雏形需要进一步修改才可以使用。

机构系统的世界坐标轴设置类似于前面介绍的虚拟现实工具箱中的设置，如图 6-49 (a) 所示，每个坐标轴的旋转方向也遵从图 6-49 (b) 中所示的右手法则。所以在平面四连杆机构仿真中，我们可以认为在该系统的 x,y 轴如图 6-62 所示，这样从纸面向上的方向即为 z 坐标轴的正方向，可以将该坐标系设置为世界坐标，所以这四个转动副均应该依 z 轴作正方向旋转。

现在考虑转动副参数设置，首先考虑图中转动副 C 的参数，双击该模块，则将得出如图 6-68 所示的对话框，其中 Number of sensor/actuator ports (检测/驱动端口数) 栏目的默认值为 0，因为在本问题中要求绘制出该转动副中的曲线，所以应该将其设置为 1，这样就能在该模块自动输出一个新的端口。如果想将该端口的值输出到普通 Simulink 模块，则需要从 Sensors & Actuators 模块组中复制一个 Joint Sensor 模块，并将其连接到 Revolute C 转动副新添加的端口上，再将引出的

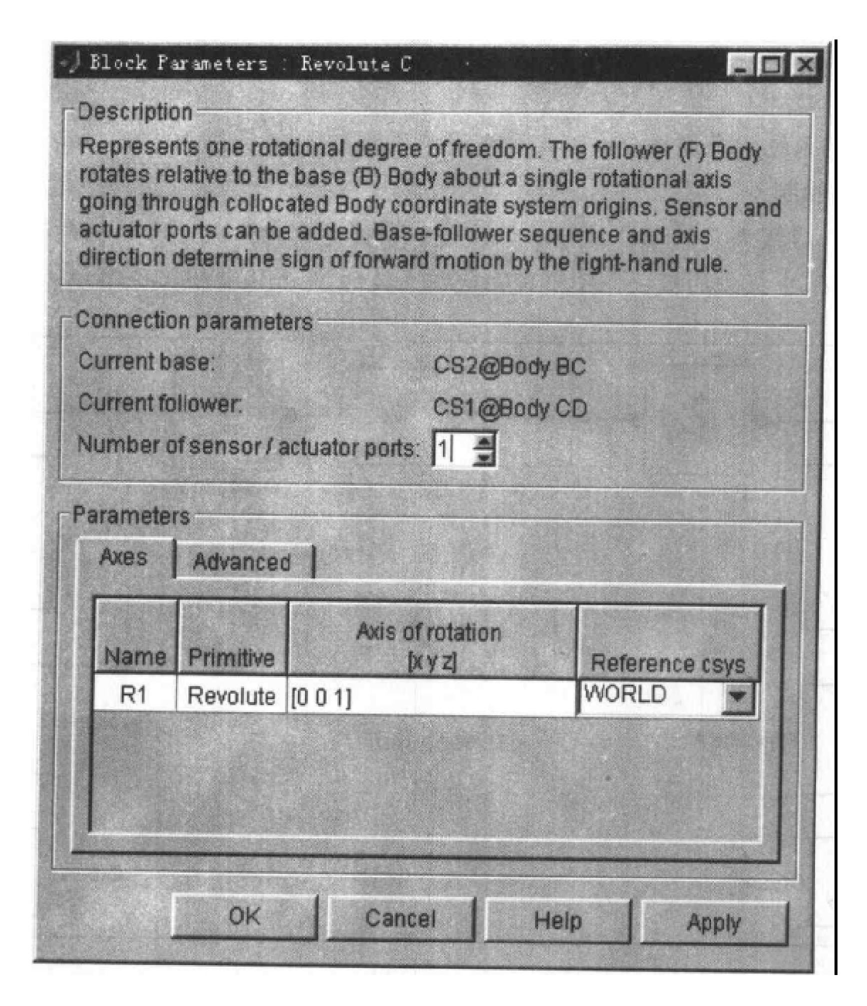


图 6-68　转动副参数设置对话框

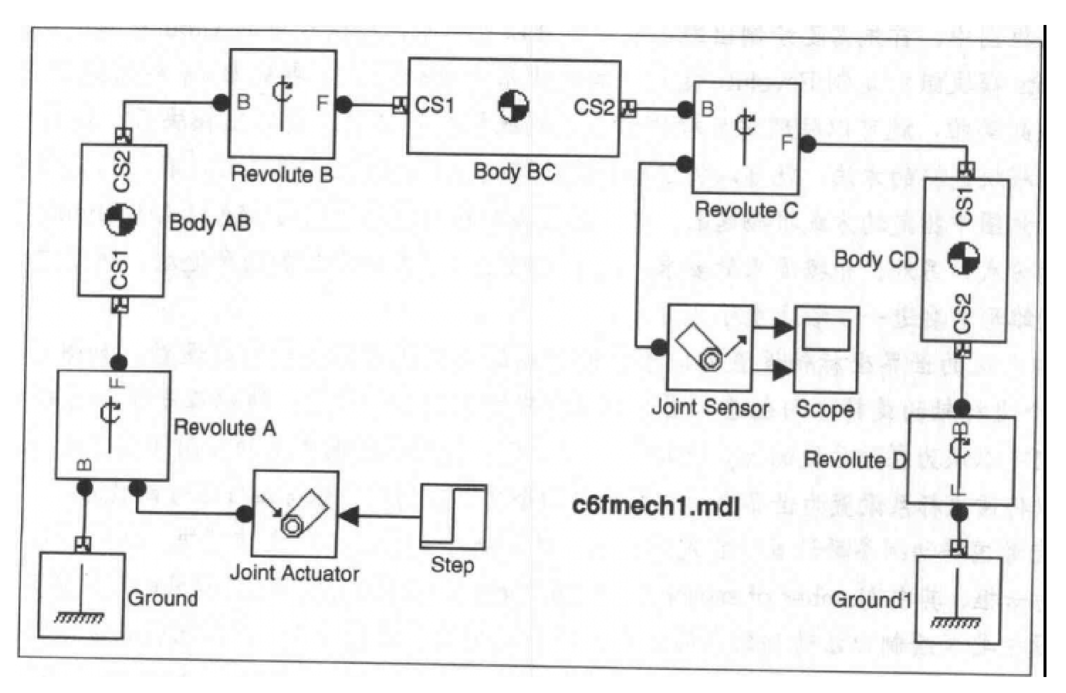


图 6-69　修改后的机构仿真框图

输出信号连接到普通示波器上即可，修改后的仿真模型如图 6-69 所示。在该仿真框图中还用同样的方式添加了转动副 Revolute A 上施加的角速度，该角速度由普通 Simulink 的阶跃信号发生模块给出。

(2) **模块参数设置**

回到图 6-68 所示的对话框，可以发现如果该模块连接好了之后，会自动填写该模块的主动端和从动端的名称，例如在 Revolute C 模块中，主动端 (Current base) 连接到 BC 连杆的 CS2 端，从动端连接到 CD 连杆的 CS1 端上。

该对话框中还应该填写该模块参数 (Parameters)，对转动副来说，其参数即坐标系及转动向量。为简单起见，可以将这四个转动副的坐标系均选择为世界坐标 (World)，另外由于这是个转动副均只绕 z 轴正方向旋转，所以其方向向量应该取作 [0,0,1]。

假设 A 点为世界坐标系的坐标原点，则左侧机架的坐标为 (0,0,0)cm，右侧机架的坐标可以算出为 (30,0,0)cm。双击机架图标，则将得出如图 6-70 所示的对话框，我们可以将机架坐标填写到该对话框 Location 栏目中，并在单位列表框中选择 cm。

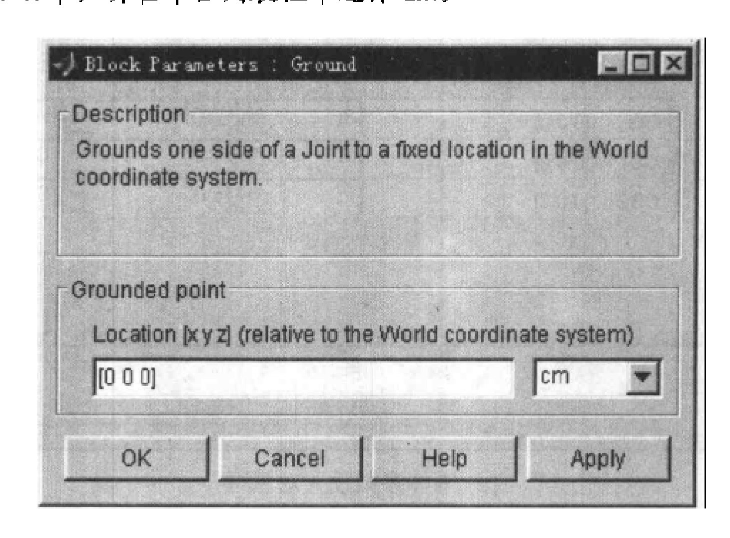


图 6-70 机架参数设置对话框

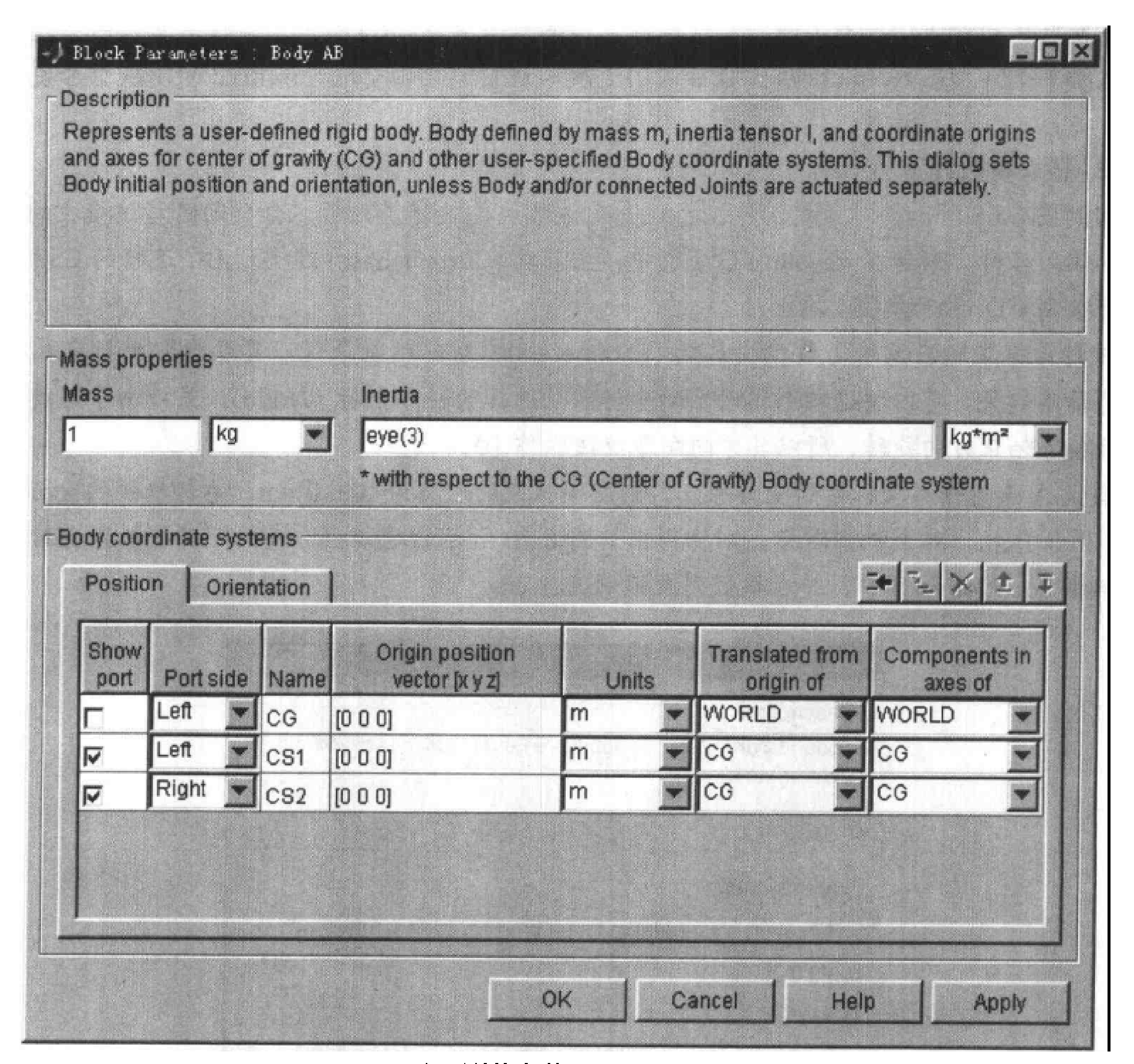
现在需要输入连杆刚体的有关参数了，双击该模块可以得出如图 6-71 所示的对话框，从该对话框中可以看出，需要输入的刚体参数为：

- **连杆长度** 虽然该对话框中并不显含长度指标，但在质量、惯性力及位置计算中都要遇到每个连杆的长度。由图 6-66 中给出的机构简图可以计算出三个连杆的长度分别为 $l_1 = 10\text{cm}$，$l_2 = 10\sqrt{2}\text{cm}$ 和 $l_3 = 20\sqrt{2}\text{cm}$。

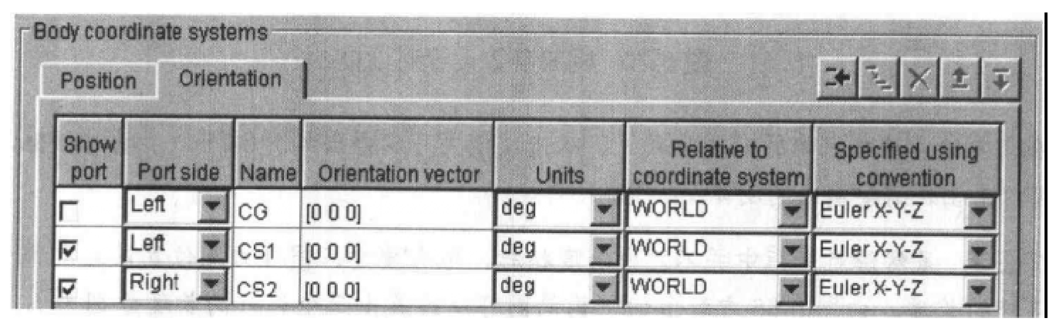
- **刚体质量** (Mass) 假设各个连杆都是均匀的圆柱形铁杆，直径为 1cm，这样，单位长度上的质量为 $7.8\pi r^2 = 6.13\text{g/cm}$。因为连杆长度都已知，所以可以立即求出各个连杆的质量。

  连杆的惯量矩阵 (inertia tensor) 是需要输入的另一个重要参数，对圆杆来说，该矩阵为对角矩阵

$$\boldsymbol{T} = \begin{bmatrix} mr^2/2 & & \\ & mL^2/12 & \\ & & mL^2/12 \end{bmatrix}$$



(a) 刚体参数设置对话框



(b) 方向设置标签

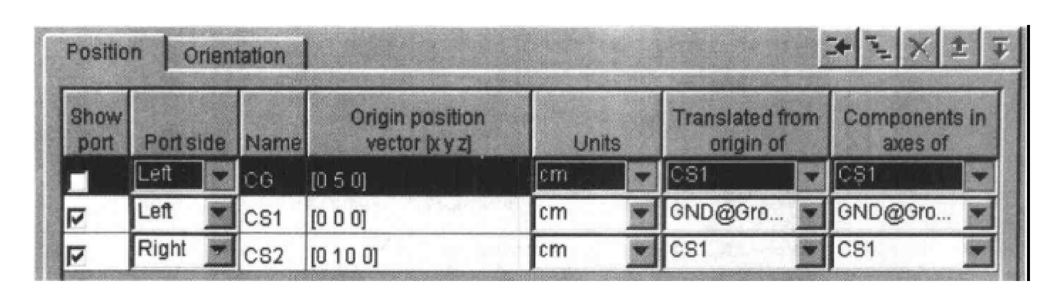
图 6-71 刚体参数设置对话框

式中，m 为杆的质量，r 为截面的半径，L 为杆长。注意，应该将其单位变换成 kg m^2。其他形状的刚体惯量矩阵计算较麻烦，通常需要求解数值积分。

- **刚体坐标系** (Coordinate system，简记为 CS) 在 SimMechanics 中描述位置时，同时还应该给出相对的坐标系，在模型 c6fmech1.mdl 中我们均采用世界坐标系。其实在机构系统研究中，

全部采用世界坐标系并非科学的方法，对不同的连杆应该采用不同的坐标系。

在这里的四连杆机构中，可以对每个连杆分别建立自己的坐标系，并以其主动端为其坐标系的原点，令连杆 AB 的坐标系为世界坐标系平移至机架 A 点所构成的 (对此连杆来说，本坐标系和世界坐标系完全一致)，则该连杆所需要填写的数据如图 6-72 所示。



Position | Orientation

Show port	Port side	Name	Origin position vector [x y z]	Units	Translated from origin of	Components in axes of
☐	Left	CG	[0 5 0]	cm	CS1	CS1
☑	Left	CS1	[0 0 0]	cm	GND@Gro...	GND@Gro...
☑	Right	CS2	[0 10 0]	cm	CS1	CS1

图 6-72　连杆 AB 的位置参数设置

对连杆 BC 来说，其坐标系可以选择为连杆 AB 的从动端 CS2 (可以理解为上一个连杆的输出端)，所以这时应该将其设置为 CS2@Body AB。这样就可以相应地填写该连杆的数据了，如图 6-73 所示。

Position | Orientation

Show port	Port side	Name	Origin position vector [x y z]	Units	Translated from origin of	Components in axes of
☐	Left	CG	[5 5 0]	cm	CS1	CS1
☑	Left	CS1	[0 0 0]	cm	CS2@Body...	CS2@Body...
☑	Right	CS2	[10 10 0]	cm	CS1	CS1

图 6-73　连杆 BC 的位置参数设置

- **连杆质心位置** (Centre of gravity，简记为 CG)　因为假设连杆是均匀的，所以连杆的质心即为连杆的中心，它们可以容易地计算出来。质心的设置仍然可以选择为世界坐标系或连杆坐标系，所以在设置上应该注意它们的设置是不同的。

综上所述，我们可以将下面参数输入到 MATLAB 工作空间，并设置出各个仿真模块的参数

```
>> r=0.5; gg=7.81*pi*r^2;
   L1=10; L2=10*sqrt(2); L3=20*sqrt(2);
   T1=diag([r^2/2, L1^2/12, L1^2/12])*L1*gg*1e-9;
   T2=diag([r^2/2, L2^2/12, L2^2/12])*L2*gg*1e-9;
   T3=diag([r^2/2, L3^2/12, L3^2/12])*L3*gg*1e-9;
```

并将相关的惯量矩阵填写到各个连杆的参数编辑框中。

(3) 仿真参数设置

在机构系统仿真中，除了设置一般的 Simulink 仿真控制参数外，还应该设置其他附加参数。安装了该模块集后，就会在 Simulink 窗口的 Simulation 菜单下添加一个 Mechanical environment...

的菜单项，选择该菜单将弹出一个如图 6-74 所示的对话框，允许用户输入输入机构系统所需的仿真控制参数。例如，在该对话框中需要输入各个坐标轴的重力加速度向量 (Gravity vector)。不难理解，对本例来说应该在该栏目中填写 [0,-9.81,0]、表示在 x 和 z 轴上的重力加速度为 0，y 轴上的为 $-9.81\ \mathrm{ms}^{-2}$。

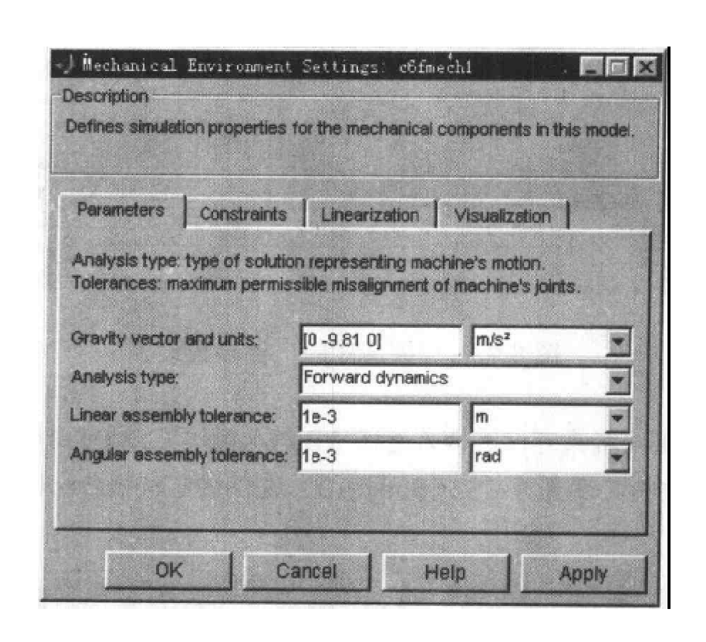


图 6-74 机构系统仿真参数设置对话框

用户还可以在该对话框中设置分析方法 (Analysis type)，如选择 Forward dynamics 则可以对系统进行前向动力学分析，亦即以从前向后的形式进行动力学分析，Backward dynamics 表示反向动力学分析，亦即从后向前的形式进行动力学分析Kinematics 表示进行静力学分析，而 Trimming 表示对原问题在平衡点处进行线性化近似分析。

仿真结果的显示也是 SimMechanics 程序的一个特色，用户既可以使用 MATLAB 自身的图形和 Simulink 的示波器 显示仿真结果，还可以依赖虚拟现实工具箱，对仿真的机构进行动画显示。单击图 6-74 中的 Visualization 标签，则将得出如图 6-75 所示的选项，可以在 Draw machine using 栏目中选择 Virtual Reality Toolbox (虚拟现实工具箱) 的显示方式。

如果选择了 Draw machine in initial state (初始时绘制机构) 选择框，则允许用户在仿真之前绘制仿真的机构动画效果，可以用虚拟现实的方式绘制机构图。

(4) **系统仿真分析** 完全设置完成后，选择 Simulation | Start 菜单，则可以启动仿真过程，得出仿真结果[①]。例如，在当前设置下可以立即得出如图 6-76 所示的仿真结果。可见，原来看起来很复杂的机构系统仿真问题利用 SimMechanics 和 MATLAB 环境可以轻而易举地解决。利用我们专门为本例制作的虚拟现实动画演示，则可以得出漂亮的虚拟现实显示，用户可以自己去理解机构系统仿真的动画设置方法。

[①] *SimMechanics* 似乎和中文 Windows 环境或相关设置冲突，不能正常运行，如果改用西文 Windows 环境，则可以正常运行仿真程序。

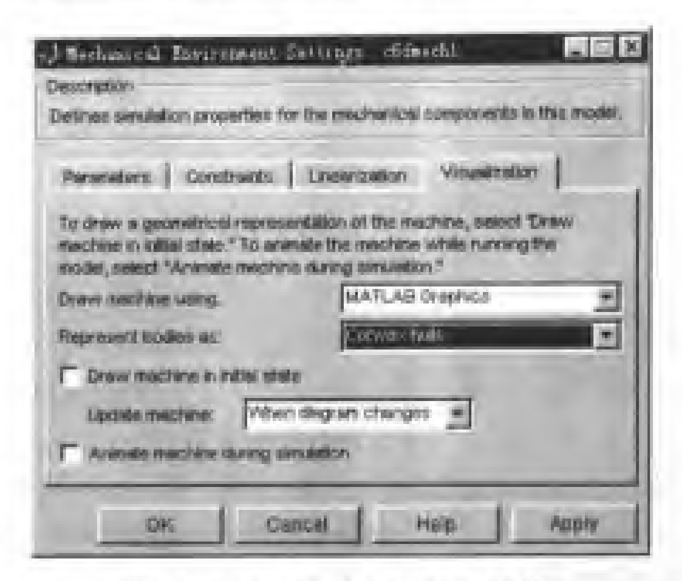


图 6-75 仿真显示方式设置对话框

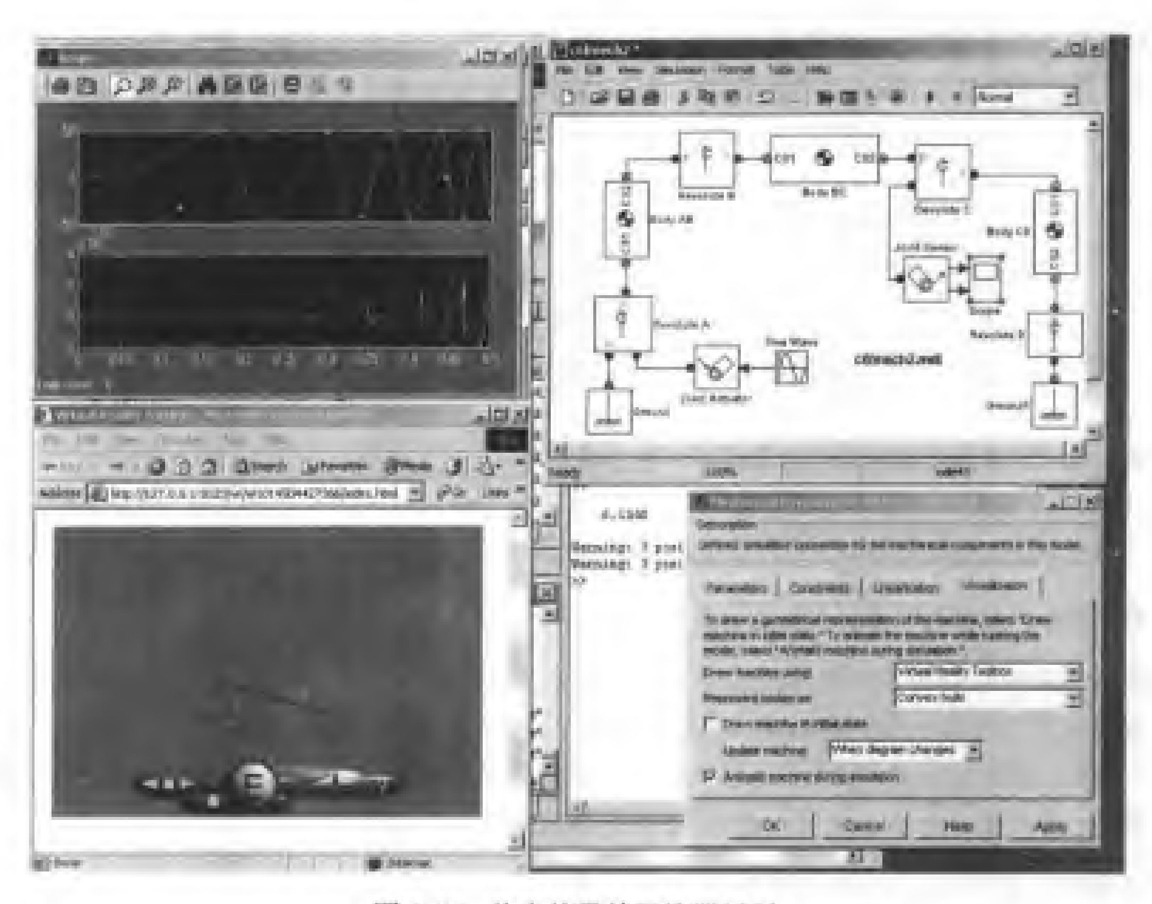


图 6-76 仿真结果的示波器显示

6.7 习 题

(1) 已知一级倒立摆的数学模型为

$$\begin{cases} \ddot{y} = \dfrac{f/m + l\theta^2 \sin\theta - \mathrm{g}\sin\theta\cos\theta}{M/m + \sin^2\theta} \\ \ddot{\theta} = \dfrac{-f\cos\theta/m + (M+m)\mathrm{g}\sin\theta/m - l\theta^2\sin\theta\cos\theta}{l(M/m + \sin^2\theta)} \end{cases}$$

其中 θ 为摆体与垂直方向的夹角 (单位为 rad), y 为小车的位移 (单位为 m), f 为电机对小车的作用力 (单位为 N), M 和 m 分别为小车和摆体的质量 (单位为 kg), l 为摆长的一半 (单位为 m), g 为重力加速度 (9.81m/s²)。试建立起倒立摆的 Simulink 模型。若取 $m = 0.21$ kg, $M = 0.455$ kg, $l = 0.61/2$ m, 并取 f 为系统的输入信号，试在平衡点 $y = \theta = 0$ 处对该系统进行线性化，并比较原系统和线性化系统的阶跃响应曲线。

(2) 假设系统的开环传递函数为

$$G(s) = \frac{1}{s^3 + a_1 s^2 + a_2 s + a_3}$$

可以按单位负反馈的方式构造出闭环系统，如图 6-77 所示．假设系统的输入为阶跃信号，则可以得出误差信号 $e(t)$, 试选定 ITAE 和 ISE 指标，分别求出这些指标下的 a_1, a_2, a_3 参数。

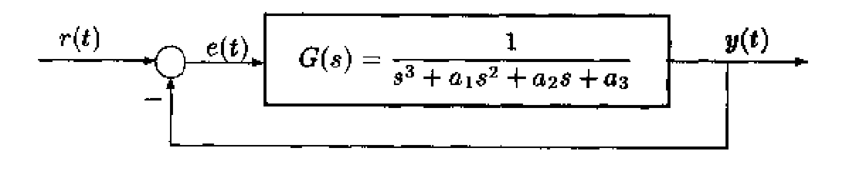


图 6-77 习题 (2) 中的单位负反馈系统

用同样的方法分别求出两个指标最小时的高阶系统

$$G(s) = \frac{1}{s^n + a_1 s^{n-1} + \cdots + a_{n-1}s + a_n}$$

的系数值。例如可以取 $n = 1, 2, 4, 5, 6$。

(3) 阅读理解用 C 语言编写的 S-函数格式，写出扩张状态观测器和自抗扰控制器的 S-函数，这些函数对应的文件名为 sfun_eso.c 和 sfun_ctrl.c。另外，在自抗扰控制器一节总共编写了 6 个 S-函数，其中三个是 MATLAB 编写的，三个是 C 语言编写的，试创建一个自抗扰控制器模块组，并将这 6 个 S-函数进行封装，置于该模块组中。

(4) 如果对象模型为不稳定的二阶模型 $1/s(s-1)$，自抗扰控制器是否还能保持其良好的控制效果．如果受控对象变成了三阶模型 $1/(s^3 + 3s^2 + 2s + 4)$，该控制器是否还有效。

(5) 例 5.19 中给出了一个系统实例，要求给带有状态变量输出的一般状态方程模型的封装模块加一个选项，使得其选中时有输出端口，没有未选中时不显示该端口．试用 C 语言编写一个 S-函数来实现这样的功能。

(6) 在第 4.6 中介绍了分形树、Mandelbrot 图和 Julia 图，这些内容都是由 MATLAB 语句编写的，试用 Simulink 的流程控制结构搭建出相应的框图，并生成这些图形结果。

(7) MATLAB 和 Spice 语言的接口函数 spice.c 由其开发者提供，试将其转换为 C 语言的 S-函数，使得以后可以将其嵌入到普通 Simulink 模型中。

注 可以选择定步长方式改写该函数，另外看看是否能找到某种方法将其中每次调用时启动和关闭 DOS 窗口的过程略去，使得仿真更连贯。

(8) 考虑大时间延迟受控对象模型

$$G(s)=\frac{10}{2s+1}\mathrm{e}^{-20s}$$

试分析用自抗扰控制器能否直接控制该模型。如果不能直接控制，则考虑调整控制器的参数，观察控制效果，以期得出合适的控制器。

(9) 考虑用自抗扰控制器是否能控制前面介绍的单级倒立摆系统，试进行仿真分析，并将结果用虚拟现实的方式显示出来。

(10) 在例 6.22 中对飞机飞行的角度做了近似，假设了飞机在 y 轴上没有位移，试考虑在 y 轴上的位移，编写更精确的飞行方向计算程序，改善虚拟现实显示效果。

(11) 试使用 SimMechanics 对习题 (1) 中的机构进行建模与仿真、并与所得出的结果进行比较。

(12) 在机构系统仿真模型 c6fmech2.mdl 中，设置了虚拟现实的动画显示环境，并进行了初始图形绘制，试结合该模块集手册[31] 理解并学会虚拟现实环境的绘制与设置，为以后可能进行的机构系统仿真打下基础。

第 7 章 半实物仿真与快速原型设计技术

在前面几章中，介绍了如何用 Simulink 进行复杂系统仿真的方法，从单变量系统到多变量系统，从连续系统到离散系统，从线性系统到非线性系统，从时不变系统到时变系统都可以用 Simulink 进行描述与仿真。引入的 S-函数可以描述更复杂的过程，而 Stateflow 技术允许利用有限状态机理论对时间驱动的系统进行仿真。

然而直到现在我们所讨论的都是纯数字的仿真方法，并未考虑和外部真实世界之间的关系。在很多实际过程中，不可能准确获得系统的数学模型，所以也就无从建立起 Simulink 所描述的框图，有时还因为实际模型的复杂性，建立起来的模型也不准确，所以需要将实际系统模型放置在仿真系统中进行仿真研究。这样的仿真经常称为“硬件在回路”(hardware-in-the-loop，简称 HIL) 的仿真，又常称为半实物仿真。因为这样的半实物仿真是针对实际过程的仿真，又是实时进行的，所以有时还称为实时 (real time，简称 RT) 仿真。

在实际控制中，半实物仿真通常有两种情况：其一是控制器用实物，而受控对象使用数字模型。这种情况多用于航空航天领域，例如导弹发射过程中，因为各种因素的考虑不可能每次发射实弹，而需要用其数字模型来模拟导弹本身的过程，这时为了测试发射台的可靠性，通常需要使用真正的发射台，从而构成半实物仿真回路。另一种半实物仿真的情况更常见于一般工业控制，可以用计算机实现其控制器，而将受控对象作为实物直接放置在仿真回路中，构造起半实物仿真的系统。在本书中所涉及到的半实物仿真局限于后一种情况。

在传统的数字仿真中，其结果的验证 (validation) 是必要的，但也是非常难以实现的。在即将推出的 Simulink 5.0 中包含了一个模型验证 (Model Verification) 模块组，其内容如图 7-1 所示。该模块组中的模块能部分解决模型验证问题，但还是只限于数学模型的验证。

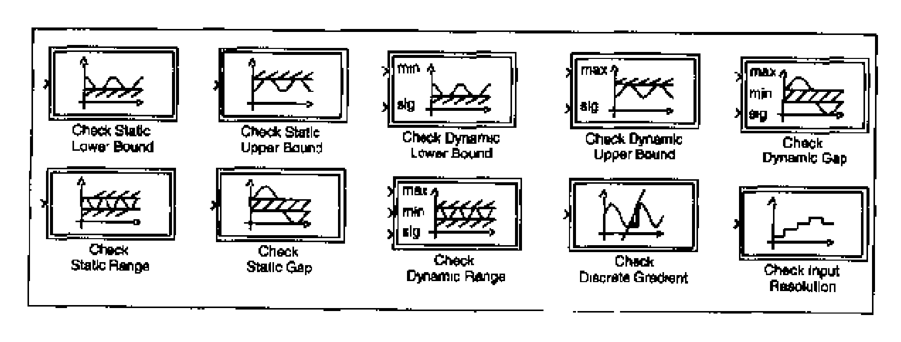


图 7-1 模型验证模块组

在实际的项目中，经常需要花大量的时间从真实的过程中收集可靠数据，与仿真结果进行比较。如果比较的结果不一致，这就需要对仿真的模型进行改进，得出的结果还需要多次往复实验，才能最终相信仿真结果。

半实物仿真的最大优势是仿真结果的验证过程是直观的，所以在一些工业过程中，采用半实物仿真的策略可以大大地缩短产品开发周期。目前应用半实物仿真最广泛的领域包括汽车制造业、硬盘驱动器开发与 CD/DVD 驱动器等产品的开发，从这方面发表的文献看，使用半实物仿真和快速原型设计技术，尤其是用基于 MATLAB/Simulink 的半实物仿真方法，大大地缩短了相关产品的开发周期，提高了产品的可靠性，有着巨大的前景。

The MathWorks 公司开发的支持 Simulink 控制器的主要有以下的工具：

- **实时工具** (Real-Time Workshop®，简称 RTW)，实时工具可以由 Simulink 的框图生成优化的语言 (如 C 和 Ada) 代码，产生的代码既可以提高仿真的速度，又可以生成半实物仿真和实时控制与快速原型设计所需的代码。RTW 建立起偏重软件的系统设计结果和偏重硬件的产品开发之间的联系，图 7-2 中描述了 RTW 在系统设计中的地位和作用[30]。

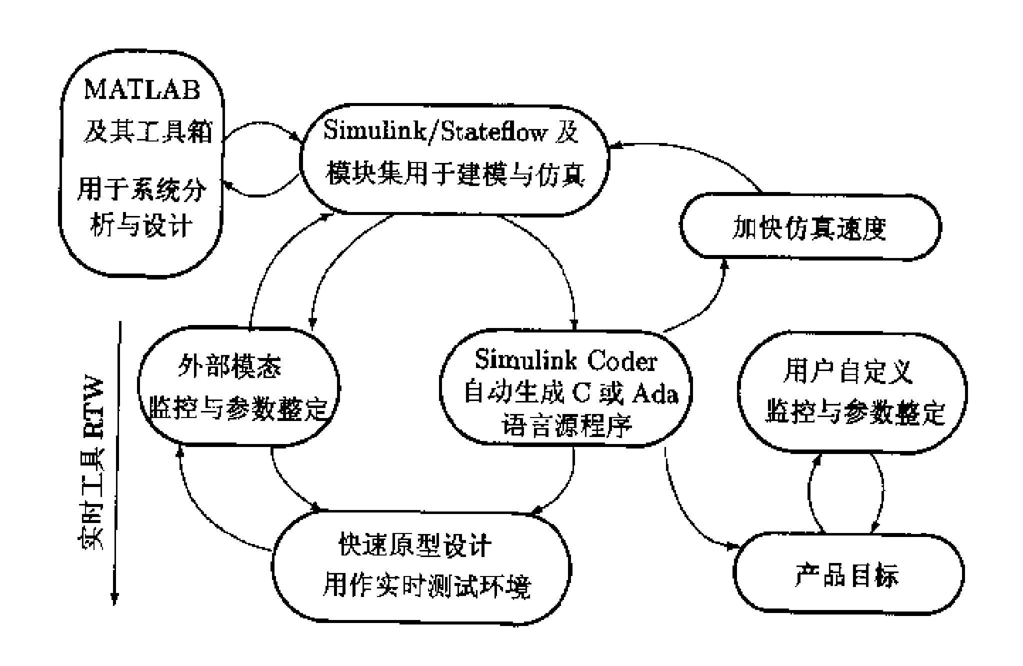


图 7-2 RTW 的地位与作用

- **实时工具嵌入式代码生成器** (Real-Time Workshop Embedded Coder®)，可以用来开发嵌入式操作系统的 C 语言程序。
- **实时工具** Windows Target®。
- xPC Windows Targets®，可以将 Simulink 描述的控制器直接通过输入输出卡 (包括在卡上 A/D 和 D/A 转换器) 对硬件系统进行实时控制。

此外，还有第三方提供的和 Simulink 的接口软硬件程序，如 dSPACE 及和它配套的 Control Desk 软件等；另外，MATLAB/Simulink 还支持很多著名的控制器厂家的产品，如 Motorola、Taxes Instrument 等，构造的仿真框图可以直接为这些控制器生成代码，更增强了其在应用方面的优势。本章将简单介绍这些工具的基本使用方法，并介绍其在半实物仿真中的应用。

7.1 Simulink 仿真的实时工具 RTW

7.1.1 独立程序生成

前面介绍过，由 Simulink 直接绘制出来的框图有时仿真速度较慢，所以可能要求加速仿真过程；另外，有时还需要使该程序能脱离 MATLAB 环境独立执行，所以在某些场合需要将其转换成可执行文件，以加快其运行速度，也可以在没有安装 MATLAB 的机器上对相应的系统进行仿真研究。

和 Simulink 下纯数字仿真不同，实时仿真需要选择定步长的仿真算法，所以在使用实时工具前应该进行相应的设置，具体方法见第 4 章中图 4-26。下面将通过例子演示实时工具的应用。

〖例 7.1〗 考虑例 6-38 中给出的 Simulink 模型 c6mstr1.mdl，可以选择定步长的 ode5(Dormand-Prince) 算法，并选择步长为 1e-4，另外，将赋值语句 M=0.01; K=1; Fsliding=1; Fstatic=1; 写入该模型的 PreLoadFcn 属性，将其存成新的文件 c7fstr1.mdl。运行该模型则可以得出如下的结果：

```
>> tic, [t,x,y]=sim('c7fstr1'); toc
elapsed_time =
   13.6800
```

可以看出，执行该系统的仿真需要 13.68 秒的时间。选择该模型窗口的 Tools | Real-Time Workshop | Build Model，则在 MATLAB 的命令窗口中将给出一系列编译的中间信息，并最终得出

```
### Created executable: c7fstr1.exe
C:\MATLAB6p1\work\c7fstr1_grt_rtw>
### Successful completion of Real-Time Workshop build
      procedure for model: c7fstr1
```

表明已经成功地得出了该模型相应的可执行文件 c7fstr1.exe，该文件可以脱离 MATLAB 环境直接执行，也可以在 MATLAB 命令窗口内用惊叹号 (!) 引导运行，下面运行该可执行文件。

```
>> tic, !c7fstr1
   toc % 这个语句不能加在 !c7fstr1 后，否则会被误认为附加参数
** starting the model **
** created c7fstr1.mat **
elapsed_time =
    2.7400
```

可以发现该程序执行速度明显加快了，对这个例子来说大约是原来Simulink 模型的 5 倍。该可执行文件将结果存到 c7fstr1.mat 数据文件。用 load 命令将该文件中数据调入 MATLAB 工作空间，这时将在命令空间出现 rt_tout 和 rt_yout 两个变量，分别存放时间向量和输出信号构成的矩阵。例如用下面的命令则可以绘制出仿真的结果。

```
>> load c7fstr1; plot(rt_tout,rt_yout)
```

事实上，如果不单纯追求可以独立执行的仿真程序，只想加快仿真的过程，则不一定非得取定步长仿真的算法，只需选择 Simulation | Accelerator 菜单项，则可以自动构造动态连接库文件，从而直接进行仿真，加速仿真过程。例如若将原模型另存为 c7fstr3.mdl 文件，经过编译过程，再进行仿真则可以测出实际耗时，可见仿真过程可以大大加快。

```
>> tic, [t,x,y]=sim('c7fstr3'); toc
elapsed_time =
   1.4500
```

带有 C 语言编写的 S-函数模块也可以转换成独立的可执行文件，加快运算速度，但由 MATLAB 语言编写的 S-函数则不能进行这样的转换。

7.1.2 实时仿真与目标计算机仿真

这里将着重介绍基于实时工具 Windows Target 的实时仿真技术，首先应该对其环境进行设置：

```
>> rtwintgt -install
You are going to install the Real-Time Windows Target kernel.
Do you want to proceed? [y] :
You must reboot your machine to finish the installation.
Do you want to reboot now? [n] : y
```

安装后应该重新启动系统，这样就可以使用 RTW Windows Target 了。另外，若想成功地运行 RTW Windows Target，则需要在机器上安装 Microsoft Visual C++ 5.0 以上版本或 Watcom C 10.6 以上版本的编译器。

在介绍本节之前有必要介绍“主机”(host) 与“目标计算机”(target computer) 的概念，这里所指的主机就是运行 MATLAB/Simulink 的计算机，目标计算机则是实际运行 Simulink 所生成的可执行文件的计算机，它可以通过 RS232 接口或 TCP/IP 协议与主机相连，共同完成实时仿真任务。目标程序不一定非得在目标计算机上运行，在一般的应用中有时还可以用同一台计算机完成主机和目标计算机的任务，但有些应用中，如构造 DOS 目标时，使用同一台计算机则不甚方便，因为参数调试不便。

【例 7.2】 考虑例 4.1 中给出的 Van der Pol 方程的 Simulink 框图，这里只考虑带有示波器输出的框图，如图 4-35 所示。为方便起见，将其保存为 c7fvdp1.mdl 文件，如图 7-3 所示。

在 Simulink 的模型窗口中，Simulation 菜单中给出了几种仿真状态：

- Normal (正常仿真模式) 一般可作系统的离线数字仿真研究，该模式是默认的，前面介绍的仿真均采用 Normal 仿真模式，该仿真方式是离线仿真的最常用方法。

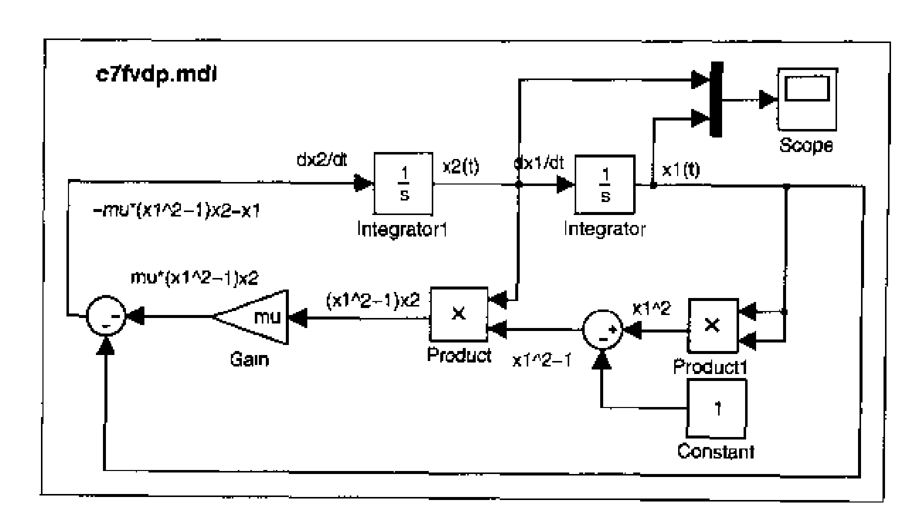


图 7-3　改写的 Van der Pol 方程框图

- Accelerator (**加速仿真模式**)　如果选择了该仿真模式，则 Simulink 会自动将 Simulink 模型自动翻译成C语言程序，并将其编译连接，生成动态连接库文件，Simulink 模型可以自动调用该 DLL 文件完成仿真过程。该仿真模式和前面介绍的独立仿真程序的生成是不同的，因为它生成的程序是不能脱离 MATLAB 环境执行的。另外，利用这样的仿真模式，可以采用变步长的仿真算法，所以它兼有速度快、算法优的特点。
- External (**外部仿真模式**)　外部模式允许在没有安装 MATLAB 的机器上直接运行装在 MATLAB 的主机上的 Simulink 程序，为获得外部函数，还应该将 Simulation | Simulation parameters 菜单弹出的对话框中 Real-Time Workshop 标签下的 Category 内容选择为 GRT code generation options (代码生成选项)，并选中其中的 External mode，如图 7-4 所示。

选择了代码生成方式后还应该选择目标代码类型，在该对话框中，单击 System target file (系统目标文件) 栏目对应的 Browse 按钮。则得出如图 7-5 所示的对话框，可以从中选择所期望的目标文件类型，例如这里选择的 Real-Time Windows Target。

从该列表框可以看出，除了支持一般的 Real-Time Windows Target 之外，还支持大量其他格式的目标形式，如：

- DOS(4GW) Real-Time Target　生成在纯 DOS 下的目标程序，该程序在 Windows 环境下的 DOS 窗口中无法运行，必须以 DOS 的形式启动计算机方能运行。
- RTW Embedded Coder　生成嵌入式操作系统的程序。
- Rapid Simulation Target　快速仿真目标程序，主要用于需要频繁进行仿真的过程，如 Monte Carlo 仿真。
- Real-Time Windows Target　实时的、在 Windows 下可以运行的目标程序。
- Tornado (VxWorks) Real-Time Target　能在 Tornado 系统下运行的实时目标程序。

设置好编译环境后，就可以采用 Tools | Real-Time Workshop | Build model 菜单项进行模型编译和连接，最终生成可执行文件 c7fvdp1.exe。

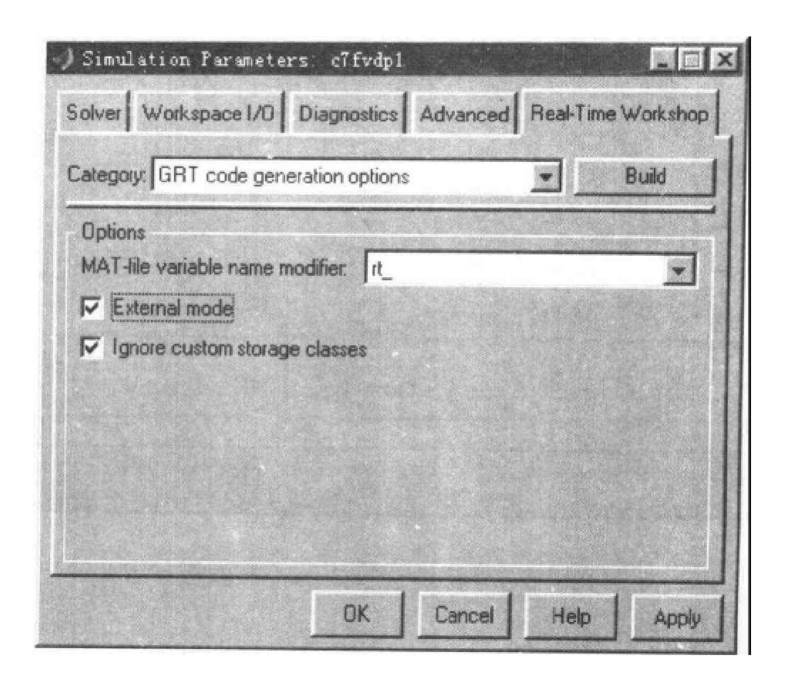


图 7-4 参数设置对话框

System Target File Browser: c7fvdp1

System target file	Description
asap2.tlc	ASAM-ASAP2 Data Definition Target
drt.tlc	DOS(4GW) Real-Time Target
ert.tlc	RTW Embedded Coder
ert.tlc	Visual C/C++ Project Makefile only for the RTW Embedded Coder
grt.tlc	Generic Real-Time Target
grt.tlc	Visual C/C++ Project Makefile only for the "grt" target
grt_malloc.tlc	Generic Real-Time Target with dynamic memory allocation
grt_malloc.tlc	Visual C/C++ Project Makefile only for the "grt_malloc" target
osek_leo.tlc	(Beta) LE/O (Lynx-Embedded OSEK) Real-Time Target
rsim.tlc	Rapid Simulation Target
rt_ada83.tlc	Ada83 Target for GNAT
rt_ada_sim.tlc	Ada Simulation Target for GNAT
rt_ada_tasking.tlc	Ada Multitasking Real-Time Target for GNAT
rtwin.tlc	Real-Time Windows Target
rtwsfcn.tlc	S-function Target
ti_c6701evm.tlc	Target for C6701 EVM
tornado.tlc	Tornado (VxWorks) Real-Time Target
xpctarget.tlc	xPC Target

Selection: C:\MATLAB6P1\rtw\c\rsim\rsim.tlc

OK Cancel

图 7-5 实时工具目标选择对话框

选择 Tools | External mode control panel 菜单项，则可以得出如图 7-6 所示的对话框，单击 Connect 按钮则可以在该对话框下实时运行输出该可执行文件。

用类似的方法还可以生成 DOS 的目标程序，但要求使用 DOS 版本的 WATCOM C

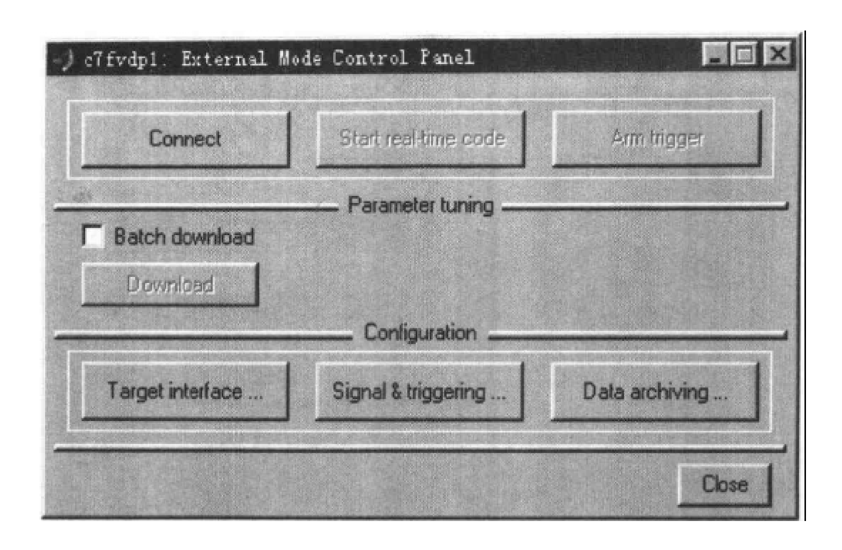


图 7-6 外部运行模式的控制面板

对程序进行编译、连接，生成的程序可以在 DOS 环境下直接执行。

7.2 xPC 在仿真和快速原型设计中的应用

7.2.1 xPC 环境简介

xPC Target 是随着 MATLAB 5.3 版本推出的，其目的是引入一种快速原型设计 (rapid prototyping) 的方法，用于控制器的实时测试和开发。它允许使用多种通用的 PC 输入输出卡，如一般台式计算机、工控机、笔记本、PC/104, PC/104+ 等用的输入输出卡；另外，它在代码翻译和编译过程中需要 C 语言编译器，如 Microsoft Visual C++ 或 MATCOM C/C++ 的支持，只有在这些编译器下才能生成独立的可执行文件，用于实时控制。

这里所谓的快速原型设计是指可以用 Simulink/Stateflow 等设计出来的控制器直接去控制受控对象实物，通过半实物仿真过程观察控制效果。如果控制效果不理想，则可以在 Simulink/Stateflow 级上调整控制器的结构或调试控制器参数，直至获得满意的控制效果。这样调试好的控制器可以认为是实际控制器的雏形 (prototype)，通过控制器设计的方法会将这样的雏形直接生成控制器，这样的控制器在开发和设计上往往能大大缩短控制器设计的过程。如果设计出来的控制器效果是理想的，还可以将其生成的 C 语言程序直接下装到控制用计算机上，还可以生成嵌入式控制器所用的控制程序，可以脱离 MATLAB/Simulink 环境直接用于实时控制，最终实现定型产品化。

xPC Target 的主要特色包括：

- 可以在没有安装 Microsoft Windows 的目标计算机上运行 Simulink 及其实时工具生成的代码；
- 最高的采样速率可以达到 100 KHz，当然这取决于处理器本身的性能；
- 支持各种各样的常用标准输入输出设备；

- 允许在主机或目标计算机上进行交互式的参数调试;
- 在主机和目标计算机上交互显示数据和信号;
- 支持通过 RS232 接口或 TCP/IP 协议的主机与目标计算机通信方式 (可以直接连接、通过局域网或互联网进行控制);
- 可以利用一般的台式机、笔记本电脑、工控机、PC/104、PC/104+、单板机、单片机、CompactPCI 等作为目标计算机进行实时控制;
- 可以用 xPC 的 Target Embedded 选项开发嵌入式控制器。这里嵌入式控制器有两种运行模式:DOSLoader 模式和 StandAlone 模式,前者用软盘以外的驱动器启动系统,并和主机相连,后者用软盘启动操作系统,并在软盘上运行内核与应用程序,在这种运行模式下应用程序可以完全脱离主机运行。

可以在主机上运行 MATLAB,Simulink 等高级语言程序,并用 C 语言编译器作为开发工具,就可以开发出实时应用程序了,如果想运行这样的实时程序,应该用一个含有 xPC Target 实时内核的特殊启动盘启动目标计算机,目标计算机启动起来后,就可以将生成的实时应用程序下装到该计算机上运行。

7.2.2 建立基于 DOS 的可执行文件

用户可以在 MATLAB 提示符下键入 xpcsetup 命令,这样就能够得出如图 7-7 所示的对话框, 在对话框中可以选择各种设置方法。在 C 语言编译器栏目可以选择 Visual C++,并按照下面的方式设置其路径:

```
CCompiler    VisualC
CompilerPath  c:\program files\microsoft visual studio
```

用户还可以使用 Sybase 公司提供的 Watcom C/C++ 编译器。该编译器 11.0c 版可以从下面的网址上免费下载:

```
http://hp.openwatcom.org
```

选定了编译器后,则应该运行 mex -setup 命令,使得编译程序的设置与选定的保持一致,这样才能正确编译所需的模型。

在该对话框中可以设置内存大小、通信方式、TCP/IP 属性。用户可以设置在 DOS 下使用的示波器,用来观测输出信号。另外,还可以选择嵌入式系统的启动方式 (xPC Target Embedded Options),从中可以选择 BootFloppy 选项,再单击 BootDisk (驱动软盘) 按钮,可以得出如图 7-8 所示的对话框,提示用户建立起可以直接驱动计算机的软盘。

建立了启动盘后,将 xPC Target Embedded Options 栏目内容设置成 StandAlone,先后单击 Update 和 Close 两个按钮,关闭该窗口,再选择 Simulink 模型窗口中的 Tools | Real-Time | Build 菜单项,则可以建立能独立执行的系统模型了。

〖例 7.3〗 下面将通过如图 7-9 所示的 Van der Pol 方程的框图来演示如何构造独立的 DOS 程序。该框图表面形式和图 7-3 中所示的完全一致,但为实现实时仿真目的,在模型仿真控制参数对话框中一定要选中如下的选项:

图 7-7 xPC Target 设置对话框

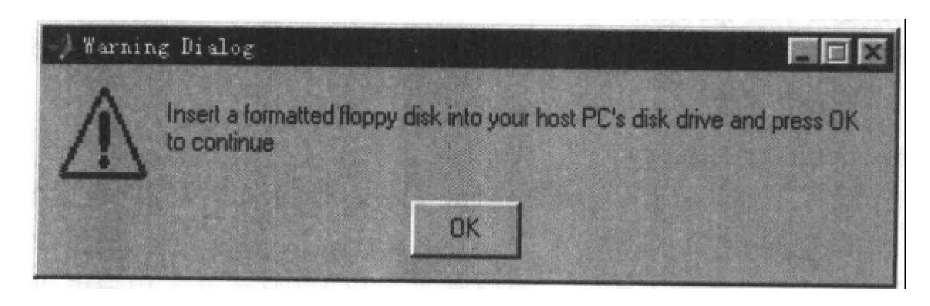


图 7-8 驱动软盘制作提示对话框

- 仿真模式可以设置为 Normal。
- 仿真参数对话框的 Solver 标签中应该选择定步长算法，并指定仿真步长的值，且从列表中适当地选择具体的微分方程求解算法。
- 在 Realtime Workshop 标签下，首先单击 Browse 按钮，在得出的列表中选择 xpctarget.tlc xPC Target 选项。
- 在该标签下再按下 Category 列表框，从中选择 xPC target code generation options 选项，单击

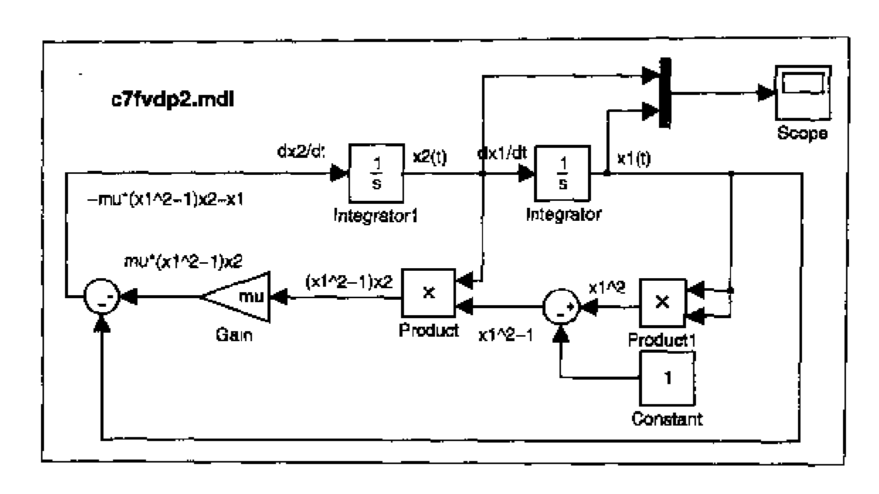


图 7-9 演示模型的 Simulink 框图

OK 按钮则完成设置。

- 选中 Tools | Real-Time Workshop | Build 菜单项，则可以建立起所需的可执行文件。正确建立起可执行文件后，则将得出如下所示的提示：

```
### Created DLL c7fvdp2.dll
    RTTarget-32 DLM Processor 3.05 (c) 1996,2000 On Time Informatik GmbH
    ### Created DLM ..\c7fvdp2.dlm
    ### xPC Target StandAlone application c7fvdp2.rtb in directory
        ..\c7fvdp2_xpc_emb created
### Successful completion of xPC Target build procedure for model: c7fvdp2
```

利用这样建立起来的软盘可以直接启动计算机，并调入 xPC Target 内核，使得生成的可执行文件在 DOS 下直接执行。在启动过程中首先显示：

```
xPC Target 1.2 32-Bit Boot Code (c) 1996-2001 The MathWorks, Inc
Loading kernel ............................................
```

启动起来后就可以显示如图 7-10 所示的界面。

7.2.3 基于 xPC 的半实物仿真技术

在一般的离线控制系统设计中，通常需要提取出受控对象的数学模型，然后根据受控对象的数学模型来设计控制器。在仿真框图中往往使用的是受控对象的数学模型，所以仿真结果是针对数学模型得出的纯数字结果，如果将这样设计出的控制器进行硬件实现，直接用于实际受控对象的控制，就不一定能得出满意的控制效果了，因为在基于模型的纯数字仿真中忽略了很多因素，例如模型的准确性、外部扰动、模型本身的参数变化和结构变化、检测信号的量测噪声等，所以即使纯数字仿真能得出理想的结果，将其用于实际系统也可能走样。

这样，半实物仿真技术就显得十分重要了，因为设计出来的控制器可以直接对实际受控对象进行控制，所以可以立即得出对其控制效果的评价。xPC 是一种较理想的廉价

```
Loaded App   c7fvdp2
Memory   1MB
Mode RT, single
Logging   t,x,y
Stop time   20
Sample time   0.001
Average Tet:   -
Execution:   stopped
```

```
-------------------------------------------------------
xPC Target 1.2 (c) 1996-2001 The MathWorks, Inc *
-------------------------------------------------------
System:  Host-Target Interface is RS232 (COM 1/2)
System:  COM 1 detected
System:  download started ...
System:  download finished
System:  initializing applications...
```

图像显示窗口

图 7-10 独立 DOS 程序显示内容

半实物仿真系统，因为它支持很多常用的输入输出接口卡，所以可以将实际受控对象通过接口卡和计算机连接起来，进行仿真分析，观察控制效果。

在 MATLAB 命令窗口中键入 `xpclib` 或在 Simulink 模块库下双击 xPC Target 模块组图标，则得出如图 7-11 所示的 xPC 模块组，可见在该模块组中有各种常用的硬件设备图标，例如模数转换器 (A/D)、数模转换器 (D/A)、计数器 (Counter) 等，故用户可以根据自己系统的设置选择合适的设备。

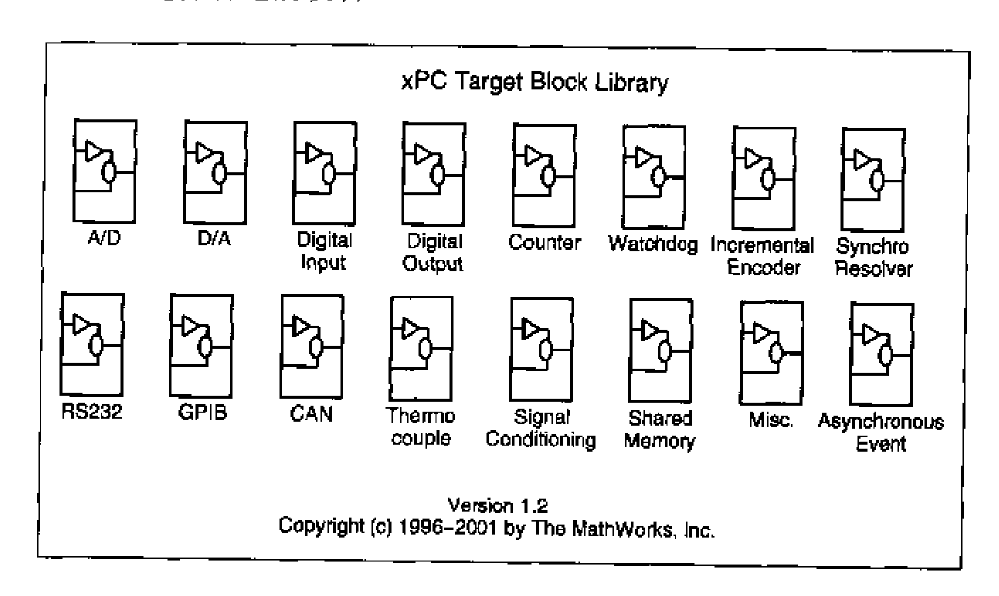


图 7-11 xPC Target 模块组

例如，用户可以双击其中的 A/D 图标，这样就能得出如图 7-12 所示的模数转换器组，其中给出了几乎所有著名的 A/D 转换器研制者的名称，双击其中的 Advantech (研华) 图标，则将进一步得出如图 7-13 所示的研华 A/D 设备，用户可以从中选择适当的模数转换器名称，连到 Simulink 构造的控制器的输入端即可。

还可以通过相似的方法选择合适的数模转换器，将之连接到 Simulink 搭建的控制器

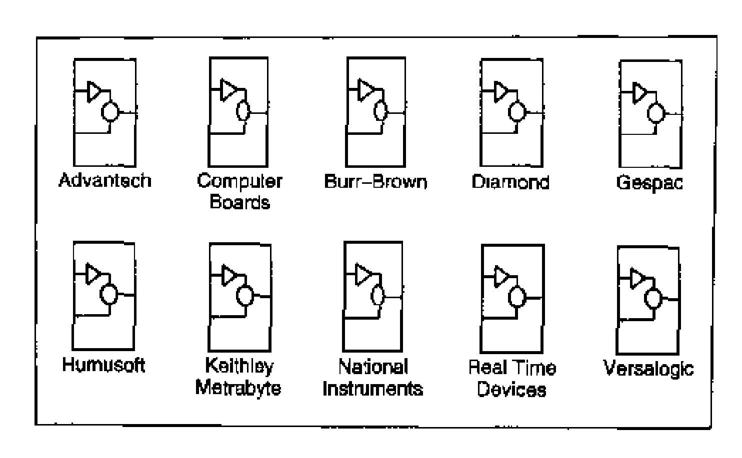


图 7-12 xPC 支持的模数转换器

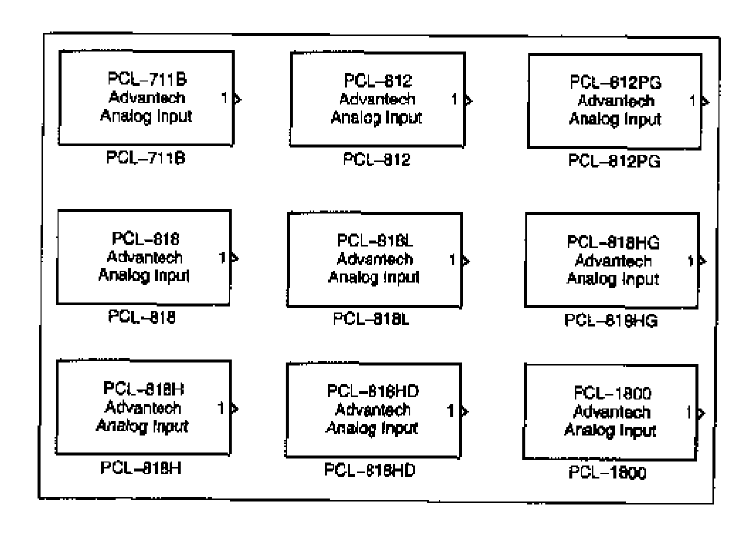


图 7-13 研华模数转换器

的输出端。这样，将受控对象实物的输入信号和输出信号分别接到插入计算机的输入输出板的输入端和输出端，就可以构造出半实物仿真的结构图，可以进行控制器的调试。

这里主要以研华输入输出卡 PCL1800 为例来介绍 xPC 的应用。PCL1800 是带有 ISA 插槽的输入输出卡，可以直接插到计算机主板上，完成其输入与输出功能。该卡带有 16 路 12 位的模拟通道、最大允许的采样速率为 330kHz，有两路 12 位模拟 D/A 输出通道，16 路数字输入和 16 路数字输出。

〖例 7.4〗 考虑例 5.12 中给出的 PI 控制系统框图，在该系统中，可以根据受控对象的数学模型进行 PI 控制器设计，并得出如图 7-14 所示的 Simulink 框图。

前面指出，这样的仿真框图是对受控对象的数学模型在控制器作用下的数字仿真，其仿真结果不一定和实际受控对象在该 PI 控制器作用下的效果一致，甚至它们之间将有很大的偏差。如果想测试实际受控对象在该控制器下的控制效果，则需要首先从 Simulink 框图中删除原始的数学模型，将控制器的输出连接到 xPC 提供的 D/A 转换器端口（如这里采用的研华 PCL 1800），并将系统输出的反馈信号和 xPC 提供的 A/D 转换器相连，得出如图 7-15 所示的结果。

得出了框图之后，就可以将受控对象实物通过研华 PCL 1800 和计算机连接起来，

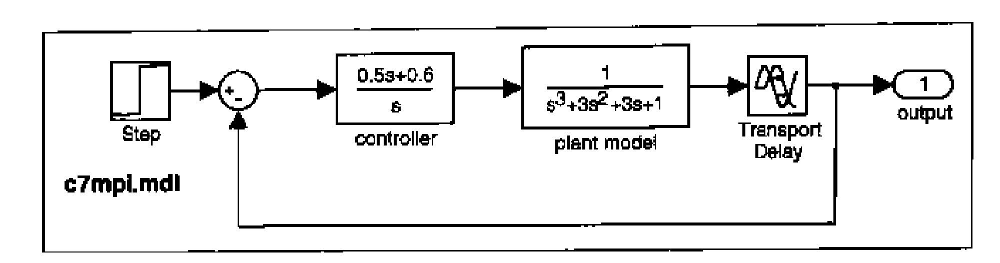


图 7-14　PI 控制系统框图

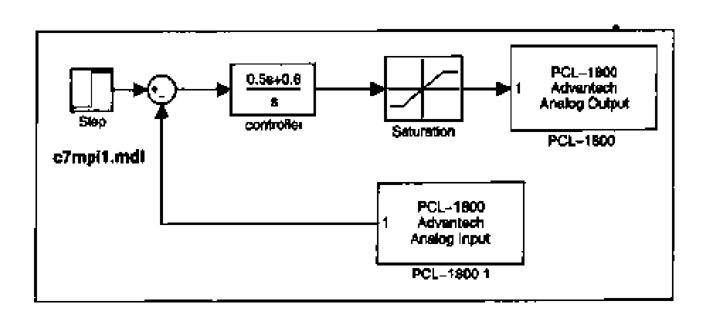


图 7-15　PI 控制系统的半实物仿真框图

即将受控对象的输入端连接到输入输出卡的 D/A 口上，接受计算机发出的控制信号，且将受控对象的输出端连接到输入输出卡的 A/D 口上。双击框图中的 A/D 模块和 D/A 模块，都应该得出类似的如图 7-16 所示的对话框，在该对话框中需要填写信号如下信息：

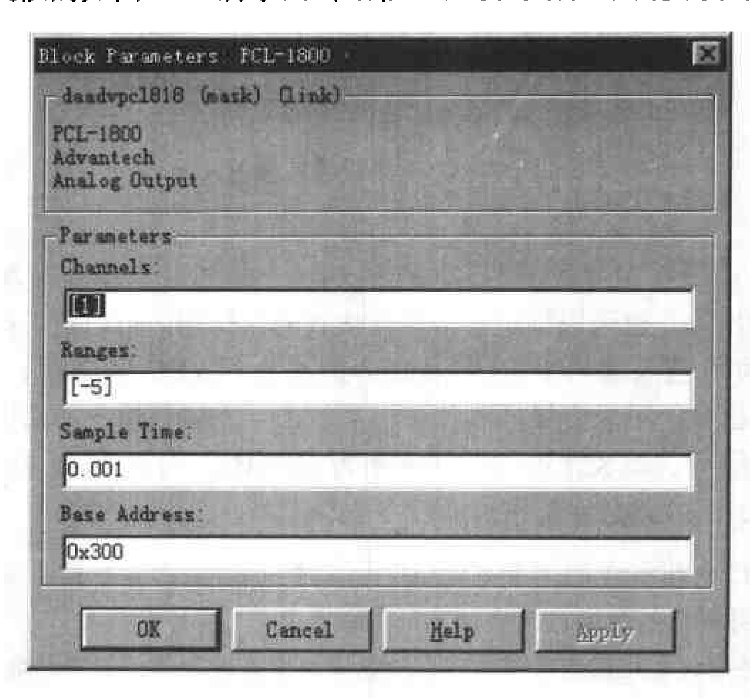


图 7-16　D/A 转换器参数设置对话框

- **信号通道** (Channels)　如果将其连接到某个通道上，则填写其序号即可，例如这里填写 1。如果进入 D/A 转换器的信号是多路的，则可以填写序号的向量。
- **信号范围代码** (Range)　在该栏目填写每个输入、输出信号的幅值范围，具体填写格

式为 [v_1, v_2, $\cdots$]，如果第 i 路信号范围为 $(-5,+5)$，则应该取 $v_i=-5$，如果该信号范围为 $(0,+5)$，则应该取 $v_i=5$。

- **采样周期** (Sampling Period) 其设置应该在 D/A、A/D 转换器的允许范围，另外应该和仿真参数中的定步长保持一致。
- **基地址** (Base Address) 该设置应该与硬件输入输出卡上的设置完全一致。

从图 7-15 中给出的系统看，该系统并不像经典的闭环系统结构，而更像一个开环系统结构，事实上，上述的连接方法可以用计算机生成的控制器直接去控制受控对象，且让系统的输出信号反馈回计算机，这事实上构成一个闭环控制系统。正是在整个的仿真回路中将实际的受控对象包含在内，所以这样的仿真结构才称为"硬件在回路"的仿真结构。

如果用户想检测出系统的输入、输出信号或中间信号，则可以将该系统改写成如图 7-17 所示的形式。这样仿真得出的信号将被存储到数据文件中，用户可以将其调入 MATLAB 工作空间，用其他方法绘制出来。如果在框图中使用示波器元件，也可以进行实时显示。

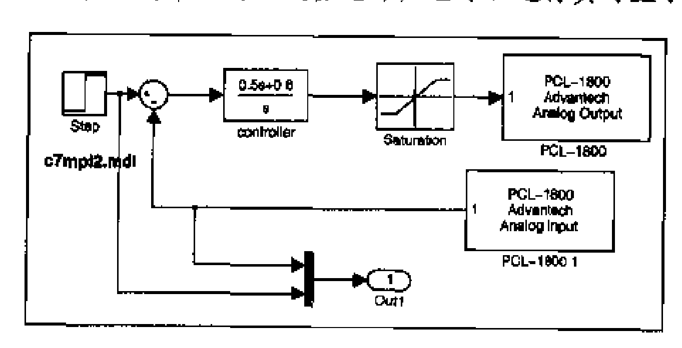


图 7-17 带有信号检测的半实物仿真框图

7.3 基于 dSPACE 的实时控制技术

前面介绍过，在控制器开发和快速原型设计中，半实物仿真技术是相当重要的，对实际系统设计出控制器的效果好坏可以直接通过半实物仿真的方法来检验。

在上节中简单介绍了基于 xPC 和研华 PCL 1800 的半实物仿真的概念，总的说来，该方法是一种低成本的解决方案，适用于实验室的一般教学实验，但不大适应于更高层次的产品开发。德国 dSPACE GmbH 公司[①]开发的 dSPACE 软硬件系统最早支持用 Simulink 直接进行硬件控制与仿真的解决方案，代表着当前国际上这方面的最高水平，适用于控制器的快速原型设计、半实物仿真、自动产品级代码生成和虚拟系统测试等，影响其在高校教学中广泛使用的主要问题可能是其不菲的价格。

7.3.1 dSPACE 硬件介绍

目前在教学和一般科学实验方面比较流行的 dSPACE 部件是 ACE 1103 和 ACE

[①] dSPACE 是 digital signal processing and control engineering 的缩写。dSPACE GmbH, Technologiepark 25, D-33100 Paderborn, Germany, (http://www.dspace.de)，其相关产品在中国国内也由恒润公司代理。

1104，包括 DSP 控制板 DS 1103 (1104)、实用控制软件 Control Desk、实时接口 RTI 和实时数据采集接口 MTRACE/MLIB，使用较方便。其中，如图 7-18 所示的 DSP 卡可以直接插入计算机的相应插槽。

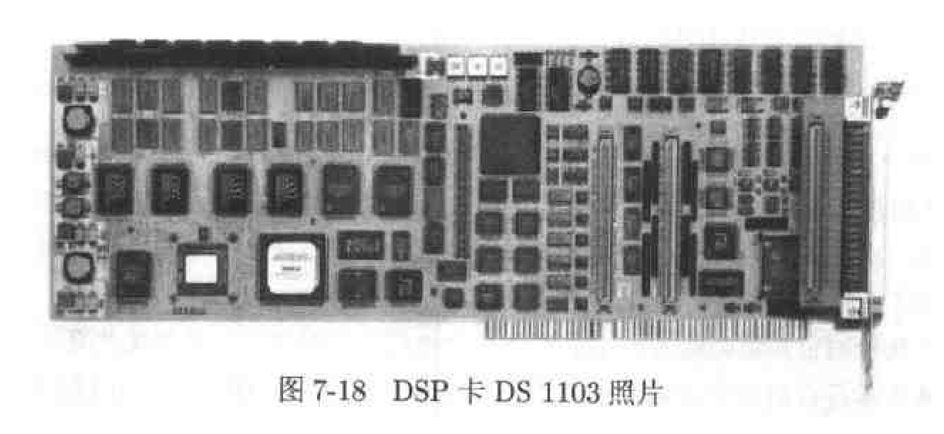


图 7-18 DSP 卡 DS 1103 照片

这里将以 DS 1103 卡为例介绍其特点[17]：

- 该卡使用的 PowerPC (333MHz) 可以直接进行浮点运算，是目前单板系统中功能最强大、输入输出点最丰富的开发系统。
- 该卡有 36 路 A/D 转换器，8 路 D/A 转换器并配有数字输出。
- 该卡集成了一个以 TI 公司的 TMS320F240 DSP 为核心的输入输出子系统，可以满足特殊的输入输出要求。例如这种 DSP 可以提供三相 PWM 信号发生器，故可以适应于拖动系统的设计。
- 集成了Siemens 的 CAN 控制器使其也可以适应汽车和自动化方面的应用。

每个板卡还带有 Simulink 实时接口库，例如在 MATLAB 命令窗口下键入 `rti` 则将打开如图 7-19 所示的 DS 1103 实时接口模块库。双击其中的 Master PPC 图标则将打开如图所示的 7-20 模块库。可以看出，在该模块库中包含大量卡上元件的图标，如 A/D 转换器，这些图标均可以拖到 Simulink 框图中，将计算机中产生的信号直接和卡上的实际信号打交道。

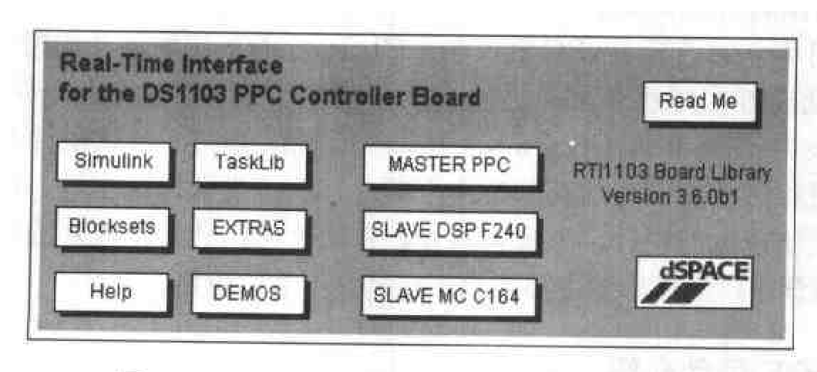


图 7-19 DS 1103 的实时接口 Simulink 模块库

【例 7.5】 VCFP (voice-coil-driven flexible plant) 受控对象是 dSPACE 公司提供的用于演示 dSPACE 产品性能的七阶线性机械装置，可以将其作为硬件受控对象进行控制。在本例中将

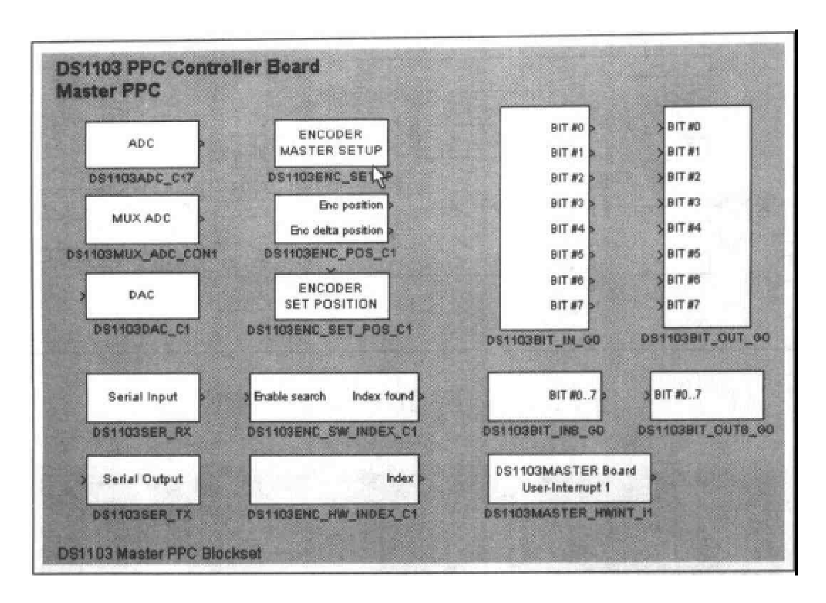


图 7-20 DS 1103 板卡上元件图标集

研究该对象的数学模型在数字式带有抗绕的 PID 控制器作用下的效果。可以容易地建立起如图 7-21 所示的纯 Simulink 仿真框图。

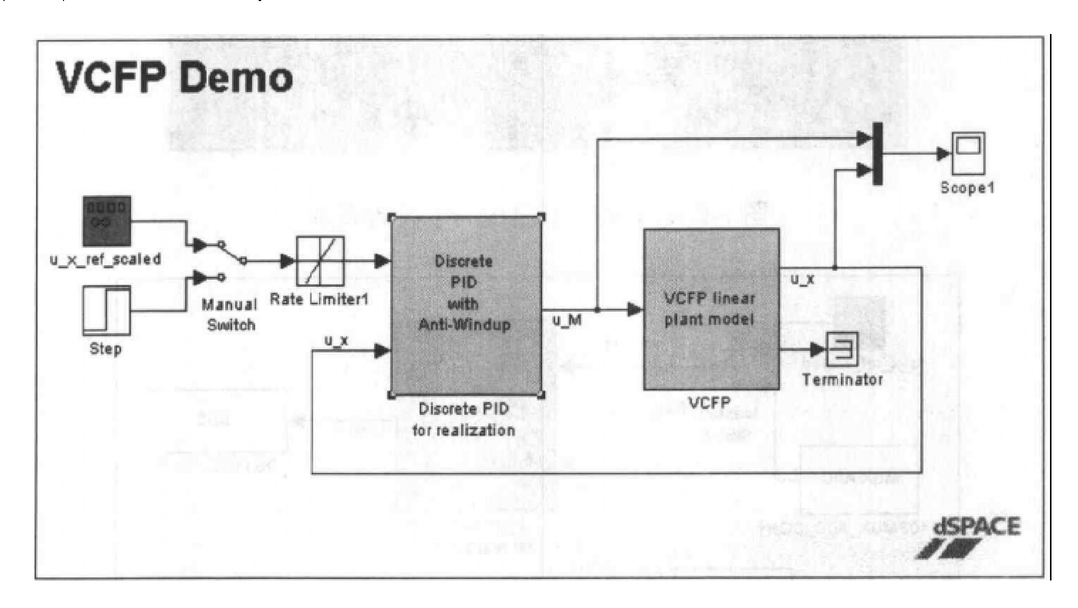


图 7-21 Simulink 仿真框图

双击其中的 PID 控制器模块，则可以立即得出该子系统模型，如图 7-22 所示。可见，在该模块下还有更下一级的子系统。

对该系统进行数字仿真，则将立即得出如图 7-23 所示的输出信号。

有了 MATLAB/Simulink 的基础知识，获得系统的纯数字仿真结果应该是相当容易的了。如果取消对象的数学模型，将受控对象实物直接连接到 dSPACE DS 1103 卡的 D/A 转换器上，并将其输出信号直接连接到其 A/D 转换器的端口上，就可以从图 7-20 所示的元件模块库中选取 A/D 转换器、D/A 转换器和信号源模块，复制到仿真模型中，得出如图 7-24 所示的仿真框图。

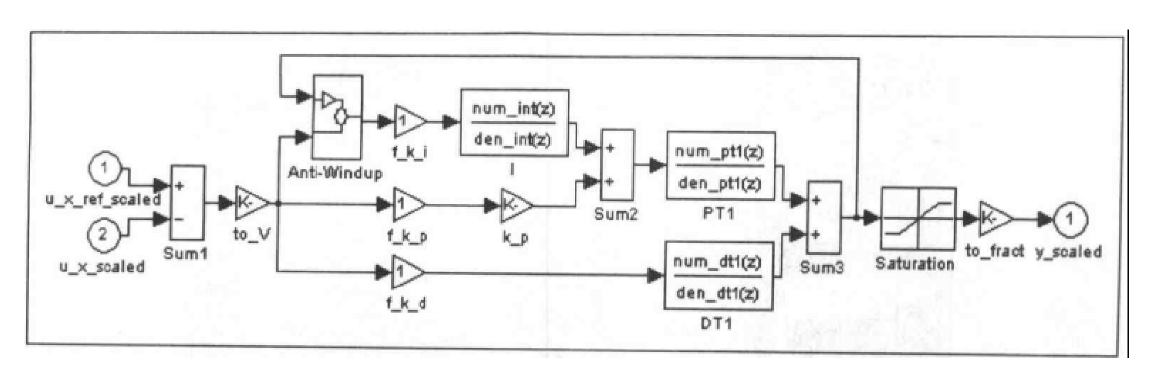


图 7-22　数字 PID 控制器子模型

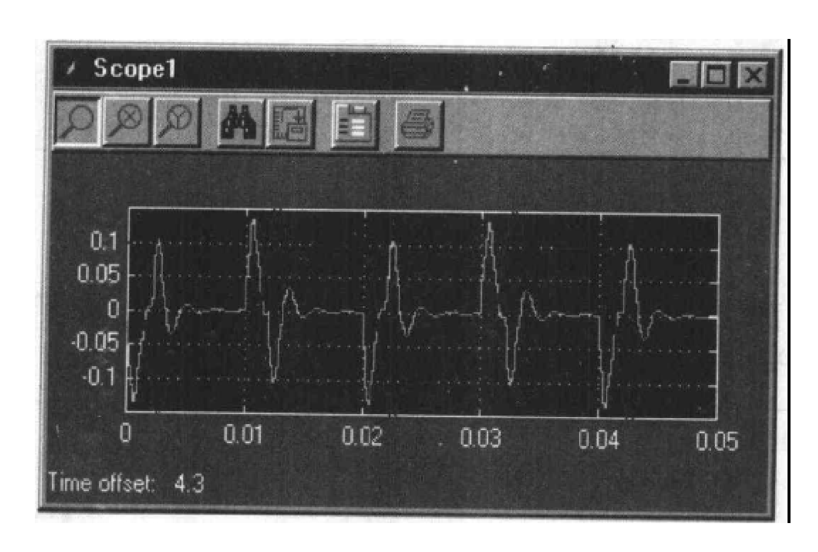


图 7-23　系统输出曲线的示波器表示

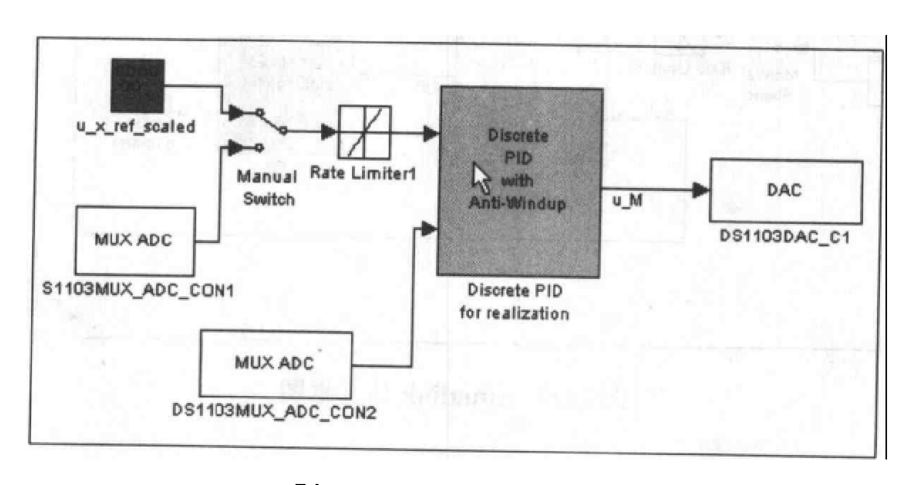


图 7-24　半实物仿真框图

7.3.2　Control Desk 及虚拟仪器开发

回忆前面介绍的 xPC 实时控制解决方案，可以发现使用和参数调试还是比较麻烦的，每次参数调试均得在主机上进行，然后才可以观测在目标机上的效果。另外，参数整定也不是很容易的。

dSPACE 系统的一个重要特色就是它提供了真正实时控制方式，它提供了一个强大的 Control Desk 软件，该软件允许用户真正实时地调整控制器参数和运行环境，并提供了各种各样的参数显示方式，适用于虚拟仪器的开发。

〖例 7.6〗 仍考虑前面的例子，启动 Control Desk 软件，再选择其中的演示起始界面 (即选择其菜单项 File | Open Experiment，并选择其中的 vcfp_demo.cdx 文件)，如图 7-25 所示。在该界

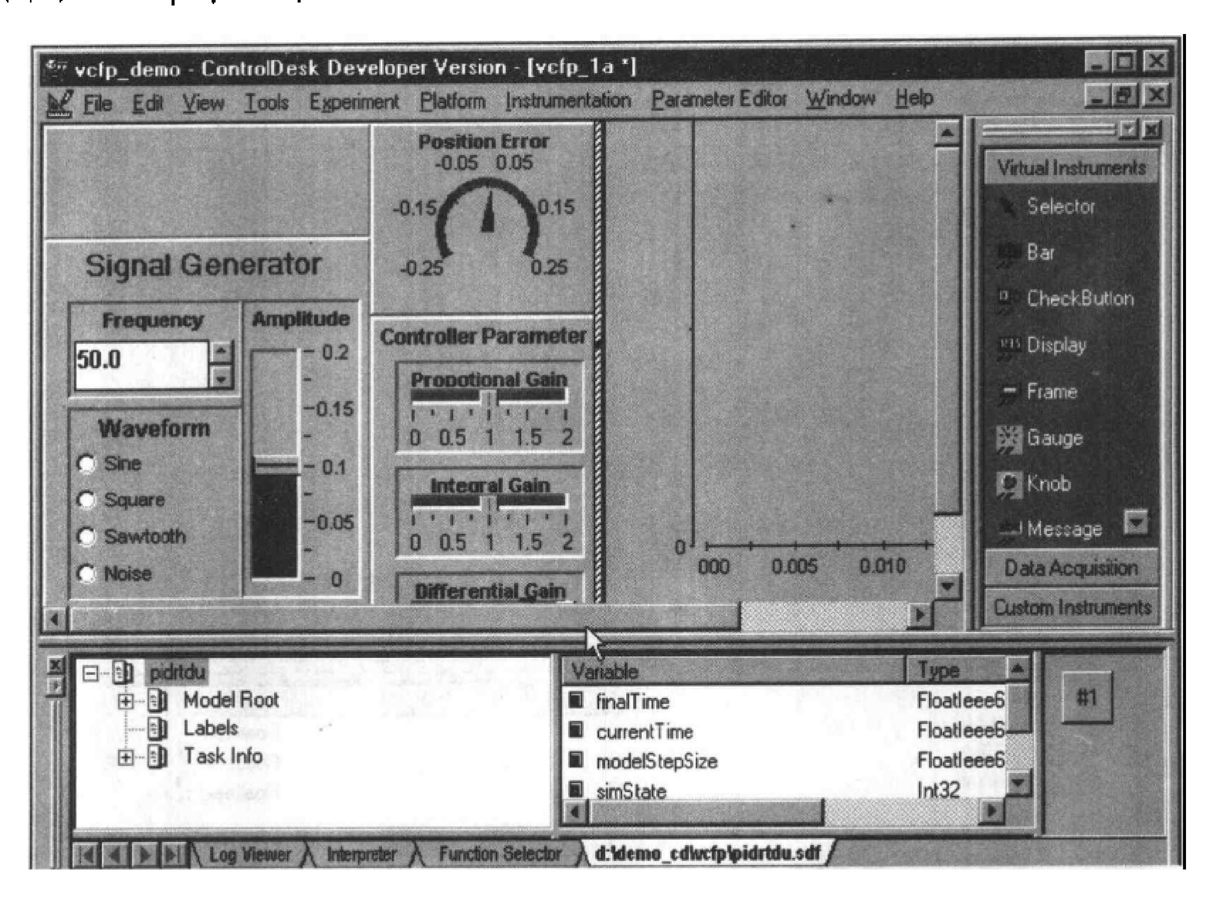


图 7-25 Control Desk 界面

面的右上角为提供的“虚拟仪表” (virtual instrument) 编辑工具栏，从中可以选择 Bar (指示条)、CheckButton (复选框)、Display (数字显示窗口)、Frame (方框) 等。例如可以选择 Display 工具，并到整个窗口的左上角空白处画一个方框，这样将在该部分建立起一个数字显示窗口。如果想在该数字显示窗口中显示当前的时间，则应该将右下角处的 variables 栏目中的 currentTime 变量拖动到该窗口上，这样就能自动在数字显示窗口上显示当前时间。

从已经建立的虚拟仪表表盘中可见，在该界面下可以显示当前时间，还可以从复选框选择系统输入信号类型，如选择正弦、方波、锯齿波和随机输入信号，可以用滚动杆设置信号的幅值和频率，还可以提高滚动杆调整 PID 控制器的 K_p, K_i 和 K_d 参数。还可以在仪表的右部设置输入输出曲线的显示。

类似地，可以充分利用其中提供的工具在窗口上显示更多的信息，建立其虚拟仪表显示系统。完成设置后，选择 Instrumentation | Animation Mode 菜单项，就可以立即用动画的方式显示仪表中的信息和当前的半实物仿真结果了。

在实时显示的过程中若将输入信号改变成锯齿波，则将立即得出锯齿波作用下的实际受控对象的响应曲线，如果调整输入信号的频率和幅值，则马上就能得到相应的响应曲线，所以这样进行系统调试的实时性远远高于前面介绍的 xPC 解决方案。

如果用户修改了 PID 控制器的 K_p 参数，则可以马上得出如图 7-26 所示的显示效果。这使得控制器的参数在线调整效果马上就能显示出来，所以用这样的方法去设计实际控制器是再简单不过的事了，利用这样的方法可以大大加快控制器的设计和调整过程。

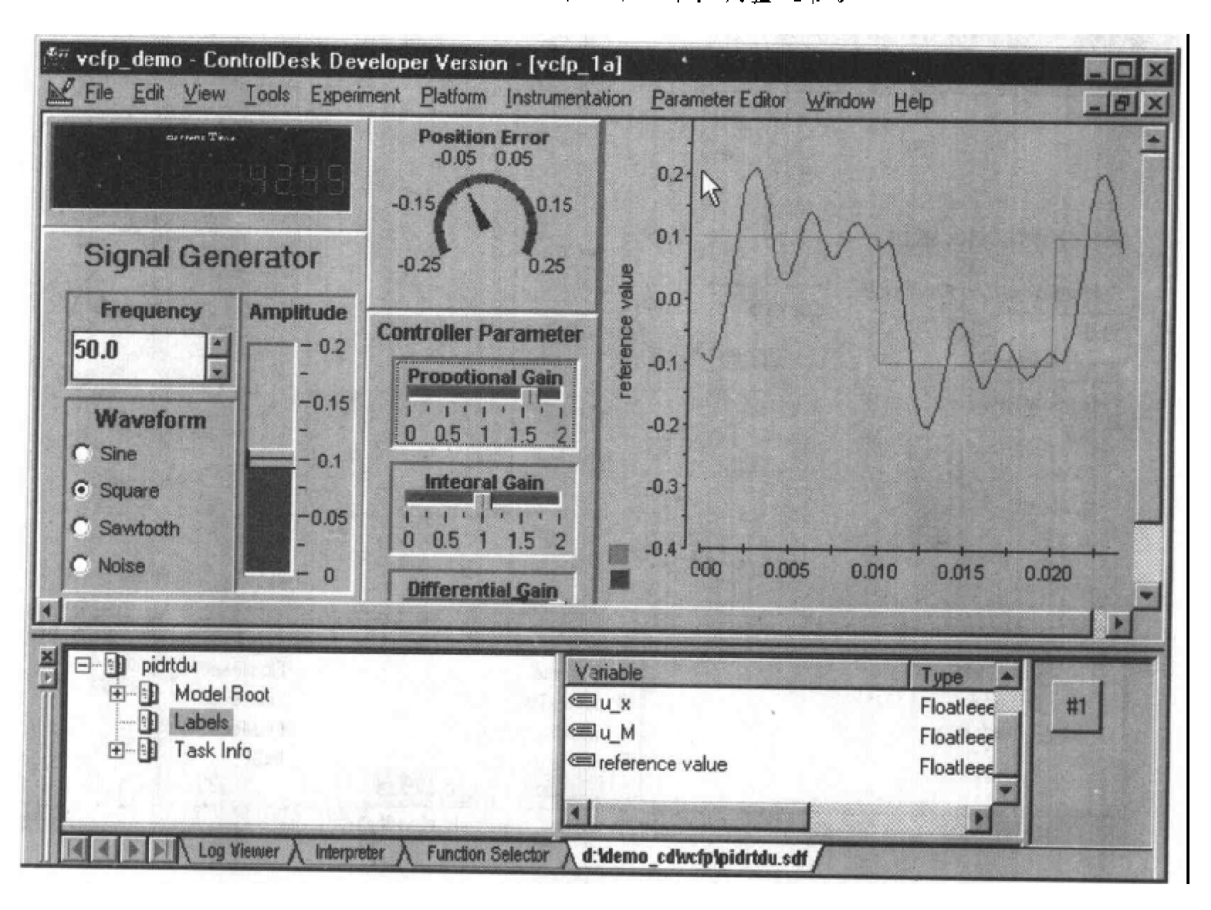


图 7-26 修改参数后立即得出的虚拟仪表显示

由于 dSPACE 提供了很高的水准，它允许用户在其中嵌入寻优程序，对控制器参数进行在线寻优，以设计出更好的控制器，所以在使用 dSPACE 时应该充分利用它本身提供的功能，更好地发挥其效益。故从工具本身看，dSPACE 是进行基于 Simulink 模型半实物仿真和实时控制的首选工具。

半实物仿真调试通过了以后，可以使用 dSPACE 提供的 TargetLink 程序将其转换成可执行文件，还可以生成更高水平的、产品级的控制程序，直接用于系统控制。

7.4 习 题

(1) 通过实例理解 Simulink 支持的几种仿真模式的概念和应用，例如其 Normal 模式、Accelerator 模式和 External 模式，并理解各种模式的适应范围。

(2) 选择以前绘制出的 Simulink 框图，试用两种方式进行实时仿真研究，如完全独立的仿真方式和主机、目标机仿真，另外还可以尝试利用互联网技术控制目标计算机，完成仿真过程。

(3) 如果有条件使用 dSPACE 等工具，自己试搭建非线性对象模型，给这个模型设计一个控制器，通过半实物仿真的方法对控制器进行有效调试，生成更高水平的、产品级的控制程序。

附录 自编的 MATLAB/Simulink 程序索引

作者为本书的各个部分精心设计了很多例子和系统框图，这些内容如果读者自行输入还是很费时间的，所以将书中涉及的作者自行编写的函数与模型放到互联网上，读者可以自行下载。相关文件可以从作者维护的"MATLAB 大观园"中下载。

http://matlab.myrice.com

在压缩的文件中包含了下面的各个文件，这些文件按出现的页码顺序排列如下：

页次	函数/文件名	类型	意　义
10	examp0.m	M 函数	例 1.3 演示函数
10	examp1.m	M 函数	例 1.4 演示函数
11	examp2.m	M 函数	例 1.5 演示函数
12	examp3.mdl	S 模型	例 1.6 演示的 Simulink 框图
12	examp4.mdl	S 模型	例 1.7 演示的 Simulink 框图
46	myhilb.m	M 函数	演示函数编写方法的程序
47	my_fact.m	M 函数	演示递归调用的求阶乘函数
48	convs.m	M 函数	多个多项式乘积函数，演示可变输入变量个数
56	teximage_c.m	M 函数	在图形上叠印 LaTeX 标识，支持中文
66	tiantan.jpg	图像	天坛图像，JPG 文件
71	c2fgui1.m	M 函数	图形界面演示函数 1，有 c2fgui1.fig 文件
77	c2fgui2.m	M 函数	图形界面演示函数 2，有 c2fgui2.fig 文件
80	c2fgui4.m	M 函数	图形界面演示函数 3，有 c2fgui4.fig 文件
81	c2fgui5.m	M 函数	图形界面演示函数 4，有 c2fgui5.fig 文件
90	mex_ex2_1.c	C 文件	例 2.31 C-Mex 文件，有 mex_ex2_1.dll 文件
91	mex_ex2_1a.c	C 文件	例 2.31 改进的 C-Mex 文件，有 mex_ex2_1a.dll 文件
92	mex_ex2_1a.m	M 函数	mex_ex2_1a.c 的帮助文件
104	poly1.m	M 函数	用 Fadeev-Fadeeva 算法求取矩阵的特征多项式
123	sinm1.m	M 函数	用幂级数展开算法求取整个矩阵的正弦
128	diffctr.m	M 函数	用中心算法求取数据数值微分的函数
131	myerrf.m	M 函数	一维被积函数
133	my2dfun.m	M 函数	二维积分的被积函数
141	rk_4.m	M 函数	定步长 4 阶 Runge-Kutta 算法求解微分方程

续表

144	lorenzeq.m	M 函数	Lorenz 方程的 MATLAB 函数描述
145	vdp_eq.m	M 函数	Van der Pol 方程的 MATLAB 函数描述,
146	vdp_eq1.m	M 函数	修改后的 Van der Pol 方程描述
147	c3ximp.m	M 函数	隐式微分方程的 MATLAB 描述
148	c3xstf1.m	M 函数	刚性方程 1 的 MATLAB 描述
150	c3exstf2.m	M 函数	刚性方程 2 的 MATLAB 描述
152	apolloeq.m	M 函数	Apollo 轨道模型的 MATLAB 描述
156	c3eqdae.m	M 函数	微分代数方程的 MATLAB 描述
156	c3eqdae1.m	M 函数	微分代数方程转换后的 MATLAB 描述
166	my2deq.m	M 函数	二元方程的 MATLAB 描述
169	opt_fun0.m	M 函数	例 3.45 中的目标函数
173	opt_fun1.m	M 函数	例 3.48 中的目标函数
173	opt_con1.m	M 函数	例 3.48 中的约束函数
174	opt_con2.m	M 函数	例 3.48 中的改进约束函数
174	opt_fun2.m	M 函数	例 3.48 中的改进目标函数
179	c3xlsq.m	M 函数	最小二乘拟合的原型函数
187	psd_estm.m	M 函数	功率谱密度估计程序
215	c4mvdp.mdl	S 模型	图 4-31 中表示的 Simulink 框图
216	c4mvdp1.mdl	S 模型	图 4-33 中表示的 Simulink 框图
217	c4mvdp2.mdl	S 模型	图 4-35 中表示的 Simulink 框图
218	c4mex2.mdl	S 模型	图 4-38 中表示的 Simulink 框图
220	c4mex3.mdl	S 模型	图 4-41 中表示的 Simulink 框图
221	c4mex4.mdl	S 模型	图 4-45 中表示的 Simulink 框图
222	c4algb.mdl	S 模型	图 4-47 (b) 中表示的 Simulink 框图
222	c4alg1b.mdl	S 模型	图 4-48 (b) 中表示的 Simulink 框图
222	c4falg1c.mdl	S 模型	图 4-48 (c) 中表示的 Simulink 框图
223	c4fdig1.mdl	S 模型	图 4-49 (b) 中表示的 Simulink 框图
228	c4mex2c.mdl	S 模型	图 4-53 中表示的 Simulink 框图
232	sc2d	M 函数	连续随机输入系统离散化函数
236	c4mrnd.mdl	S 模型	图 4-58 中表示的 Simulink 框图
237	c4mrnd1.mdl	S 模型	图 4-61 中表示的 Simulink 框图
238	c4mrnd2.mdl	S 模型	图 4-63 中表示的 Simulink 框图

续表

239	spec_fac.m	M 函数	谱分解函数
243	frac_tree.m	M 函数	分形树的计算函数
244	frac_tree1.c	C 文件	Mex 格式的分形树程序，有 frac_tree1.dll
245	julia1.m	M 函数	Julia 集的计算函数
246	julia.m	M 函数	另一种测度下的 Julia 集的计算函数
247	julia2.m	M 函数	其他映射函数下的 Julia 集的计算函数
248	mandelbrot.m	M 函数	Mandelbrot 集的计算函数
251	c4exa1a.mdl	S 模型	图 4-73 (a) 中表示的 Simulink 框图
251	c4exa1b.mdl	S 模型	图 4-73 (b) 中表示的 Simulink 框图
253	c5mvdp1.mdl	S 模型	图 5-1 中表示的 Simulink 框图
253	c5mvdp2.mdl	S 模型	图 5-2 中表示的 Simulink 框图
254	c5ex2.mdl	S 模型	图 5-3 中表示的 Simulink 框图
254	c5ex2a.mdl	S 模型	图 5-4 中表示的 Simulink 框图
255	c5ex2c.mdl	S 模型	图 5-5 中表示的 Simulink 框图
256	c5ex2f.mdl	S 模型	图 5-8 (a) 中表示的 Simulink 框图
257	c5mvdp3.mdl	S 模型	图 5-9 中表示的 Simulink 框图
257	c5mvdp4.mdl	S 模型	图 5-10 中表示的 Simulink 框图
258	c5ex2d.mdl	S 模型	图 5-11 (a) 中表示的 Simulink 框图
258	c5ex2e.mdl	S 模型	图 5-11 (b) 中表示的 Simulink 框图
259	c5fmimo.mdl	S 模型	图 5-12 (a) 中表示的 Simulink 框图
260	c5nlsin.mdl	S 模型	图 5-13 (a) 中表示的 Simulink 框图
262	c5mtab2.mdl	S 模型	图 5-18 (a) 中表示的 Simulink 框图
263	c5floop2.mdl	S 模型	图 5-21 (a) 中表示的 Simulink 框图
264	c5floop5.mdl	S 模型	图 5-23 (a) 中表示的 Simulink 框图
265	c5fmswi.mdl	S 模型	图 5-25 (a) 中表示的 Simulink 框图
266	c5mtab2d.mdl	S 模型	图 5-27 (a) 中表示的 Simulink 框图
267	c5fdae.mdl	S 模型	图 5-28 中表示的 Simulink 框图
269	c5mmore.mdl	S 模型	图 5-29 中表示的 Simulink 框图
270	c5mspc1.mdl	S 模型	图 5-31 中表示的 Simulink 框图
272	c5mitae.mdl	S 模型	图 5-35 中表示的 Simulink 框图
274	c5mstop.mdl	S 模型	图 5-38 中表示的 Simulink 框图
277	c5mdng1.mdl	S 模型	图 5-45 中表示的 Simulink 框图

续表

279	c5mdng7.mdl	S 模型	图 5-48 中表示的 Simulink 框图
279	c5mdng8.mdl	S 模型	图 5-49 中表示的 Simulink 框图
280	c5mfft.mdl	S 模型	图 5-52 (a) 中表示的 Simulink 框图
282	c5fpid1.mdl	S 模型	图 5-54 (a) 中表示的 Simulink 框图
283	c5fcon1.mdl	S 模型	图 5-55 中表示的 Simulink 框图
284	c5fcon3.mdl	S 模型	图 5-57 (a) 中表示的 Simulink 框图
285	c5fcon4.mdl	S 模型	图 5-58 中表示的 Simulink 框图
290	c5msub1.mdl	S 模型	图 5-68 (a) 中表示的 Simulink 框图
290	c5msub2.mdl	S 模型	图 5-68 (b) 中表示的 Simulink 框图
293	my_blks.mdl	S 模型	图 5-72 中表示的 Simulink 框图
293	c5f14dat.m	M 文件	F-14 战斗机模型数据赋值
294	f14sub1.mdl	S 模型	图 5-74 (a) 中表示的 Simulink 框图 (F-14 子系统 1)
295	f14sub2.mdl	S 模型	图 5-75 (a) 中表示的 Simulink 框图 (F-14 子系统 2)
295	f14sub3.mdl	S 模型	图 5-76 (a) 中表示的 Simulink 框图 (F-14 子系统 3)
296	f14sub4.mdl	S 模型	图 5-77 (a) 中表示的 Simulink 框图 (F-14 子系统 4)
296	c5f14.mdl	S 模型	图 5-78 中表示的 Simulink 框图 (F-14 整个系统)
299	c5mpow1.mdl	S 模型	图 5-84 中表示的 Simulink 框图
303	c5mmos.mdl	S 模型	图 5-90 (a) 中表示的 Simulink 框图
307	c5mmot2a.mdl	S 模型	图 5-95 (a) 中表示的 Simulink 框图
307	c5mmot2b.mdl	S 模型	图 5-96 (b) 中表示的 Simulink 框图
308	c5mmot5.mdl	S 模型	图 5-98 中表示的 Simulink 框图
309	c5ftri1.cir	Spice 文件	Spice 语言描述的电子电路模型
310	c5ftri2.cir	Spice 文件	Spice 语言描述的电子电路模型 (与 MATLAB 接口)
312	c5fncdx1.mdl	S 模型	图 5-102 中表示的 Simulink 框图
322	engine1.mdl	S 模型	图 5-117 中表示的 Simulink 框图
324	engine2.mdl	S 模型	图 5-121 中表示的 Simulink 框图
339	c6mvdp.mdl	S 模型	图 6-6 中表示的 Simulink 框图
341	c6nlsys.mdl	S 模型	图 6-8 中表示的 Simulink 框图
345	c6fdly.mdl	S 模型	图 6-9 中表示的 Simulink 框图
348	c6nlsys2.mdl	S 模型	图 6-13 中表示的 Simulink 框图
351	c6exsf1.m	S-函数	例 6.13 中的 S-函数
353	c6msf2.mdl	S 模型	图 6-15 (a) 中表示的 Simulink 框图

续表

354	han_td.m	S-函数	跟踪-微分器的 S-函数
355	ex_han1.mdl	S 模型	图 6-16 (a) 中表示的 Simulink 框图 (自抗扰控制器框图)
356	han_fun.m	S-函数	测试输入信号的 MATLAB 函数
357	han_eso.m	S-函数	扩张状态观测器的 S-函数
358	han_ctrl.m	S-函数	自抗扰控制器的 S-函数
360	ex_han2.mdl	S 模型	图 6-18 中表示的 Simulink 框图 (自抗扰控制器框图)
361	ex_han5.mdl	S 模型	图 6-20 中表示的 Simulink 框图
363	sfun_han.c	S-函数	C 格式描述的跟踪-微分器 S-函数
365	sfun_eso.c	S-函数	C 格式描述的扩张状态观测器 S-函数
365	sfun_ctr.c	S-函数	C 格式描述的自抗扰控制器 S-函数
365	ex_han3.mdl	S 模型	图 6-24 中表示的 Simulink 框图 (C 语言 S-函数)
378	c6fstr2.mdl	S 模型	图 6-43 中表示的 Simulink 框图
380	c6fstr3.mdl	S 模型	图 6-46 中表示的 Simulink 框图
379	c6fsfun.m	M 函数	描述逻辑关系的 S-函数
386	myvr1.wrl	WRL 文件	描述飞机与摩天大楼的虚拟现实文件
391	c6mvrs.mdl	S 模型	图 6-61 中给出的 Simulink 模型
395	c6fmech.mdl	S 模型	图 6-67 中给出四连杆机构 Simulink 模型
396	c6fmech1.mdl	S 模型	图 6-69 中给出修改后四连杆机构模型
403	spice.c	C-Mex	题 (7) 中的原型文件，由开发者提供
406	c7fstr1.mdl	S 模型	图 6-38 框图的定步长表示模型
407	c7fstr3.mdl	S 模型	图 6-38 框图的仿真加速模型
408	c7fvdp1.mdl	S 模型	图 7-3 框图的 Van der Pol 方程模型
413	c7fvdp2.mdl	S 模型	图 7-9 框图的 Van der Pol 方程模型

参考文献

[1] Adobe Systems Inc. PostScript language, tutorial and cookbook, Reading: Addision-Wesley Publishing Company, Inc., 1985

[2] Åström K J. Introduction to stochastic control theory. Academic Press, 1970

[3] Atherton D P. Nonlinear control engineering — describing function analysis and design. London:Van Nostrand Reinhold, 1974

[4] Atherton D P, Xue D. The analysis of feedback systems with piecewise linear nonlinearities when subjected to Gaussian inputs. In: Kozin F, Ono T (eds.). Control systems, topics on theory and application. Tokyo: Mita Press, 1991, pp23-38

[5] CAD Center. GINO-F Users' manual. 1976

[6] 蔡自兴. 机器人学. 北京：清华大学出版社，2000

[7] Crossley P R and Cook J A. A nonlinear engine model for drivetrain system development. IEE International Conference Control '91, 2:921-925, Edinburgh, U.K., 25-28 March, 1991

[8] Dongarra J J, Bunsh J R, Molor C B. LINPACK user's guide. Philadelphia: Society of Industrial and Applied Mathematics (SIAM), 1979

[9] Forsythe G E, Malcolm M A, Moler C B. Computer methods for mathematical computations. Englewood Cliffs: Prentice-Hall, 1977

[10] Forsythe G E, Moler C B. Computer solution of linear algebraic systems. Englewood Cliffs: Prentice-Hall, 1967

[11] Frederick D K and Rimer M. Benchmark problem for CACSD packages. Abstracts of the second IEEE symposium on computer-aided control system design. Santa Barbara, USA, 1985

[12] 高文焕，汪蕙. 模拟电路的计算机分析与设计 — PSpice 程序应用. 北京：清华大学出版社，1999

[13] Garbow B S, Boyle J M, Dongarra J J, Moler C B. Matrix eigensystem routines — EISPACK guide extension, Lecture notes in computer sciences. Vol. 51, New York: Springer-Verlag, 1977

[14] Gilbert J R, Moler C B, and Schreiber R. Sparse matrices in MATLAB: design and implementation. SIAM Journal on Matrix Analysis and Applications, 1992, 13: 33-356

[15] 韩京清，袁露林. 跟踪微分器的离散形式. 系统科学与数学，1999, 19(3): 268-273

[16] 贺益康. 交流电机的计算机仿真. 北京：科学出版社，1990

[17] 恒润科技. dSPACE — 基于 Windows 平台的实时快速原型及半实物仿真的一体化解决途径，2001

[18] Hydro-Québec, TEQSIM International. Power systems blockset, for use with Simulink, 2001

[19] Kahaner D, Moler C B, Nash S. Numerical Methods and Software. Englewood Cliffs: Prentice Hall, 1989

[20] 刘德贵，费景高. 动力学系统数字仿真算法. 北京: 科学出版社，2001

[21] 刘兴祥，李吉蓉，林梅等译. 机械系统的计算机辅助运动学和动力学. E. J. 豪格著. 北京：高等教育出版社，1996

[22] MathWorks. Dials & gauges blockset users' guide. version 1, 2001

[23] MathWorks. DSP blockset users' guide. version 4, 2001

[24] MathWorks. MATLAB — the language of technical computing, using MATLAB. Release 12.1, 2001

[25] MathWorks. MATLAB function reference. Release 12.1, 2001

[26] MathWorks. MATLAB release 12.1 new features, 2001

[27] MathWorks. Nonlinear control design blockset users' guide. version 1, 2001

[28] MathWorks. Optimization toolbox users' manual. Release 2.1, 2001

[29] MathWorks. Power system blockset users' guide. version 2, 2001

[30] MathWorks. Real-Time Workshop users' guide. version 4, 2001

[31] MathWorks. SimMechanics users' guide. version 1, 2001

[32] MathWorks. Simulink/Stateflow technical examples — Using Simulink and Stateflow in automotive applications. 电子版报告, 1998

[33] Melsa J L, Jones S K. Computer programs for computational assistance in the study of linear control theory. New York: McGraw-Hill, 1973

[34] Mitchell and Gauthier Associate. ACSL — Advanced Continuous Simulation Language. reference manual, 1987

[35] Moler C B, Stewar G W. An algorithm for generalized matrix eigenvalue problems. SIAM Journal of Numerical Analysis. 1973, 10: 241-256

[36] Moler C B, Van Loan C F. Nineteen dubious ways to compute the exponential of a matrix. SIAM Review. 1979, 20: 801-836

[37] Nelder J A, Mead R. A simplex method for function minimization. Computer Journal, 7: 308-313

[38] Numerical Algorithm Group. NAG FORTRAN library manual. 1982

[39] Oppenheim A V, Schafer R W. Digital signal processing. Englewood Cliffs: Prentice-Hall, 1975

[40] Press W H, Flannery B P, Teukolsky S A, and Vitterling W T. Numerical recipes, the art of scientific computing. Cambridge: Cambridge University Press, 1986

[41] 任兴权主编. 控制系统计算机仿真与辅助设计，沈阳：东北大学出版社, 1986

[42] 任兴权、薛定宇、李彦平. 控制系统仿真与计算机辅助设计. 沈阳：东北大学出版社，1996

[43] Rosenbrock H H. Computer aided control systems design. London: Academic Press, 1973

[44] Smith B T, Boyle J M, Dongarra J J *et al.* Matrix eigensystem routines - EISPACK guide, Lecture notes in computer sciences. Vol. 6 (Second edition). New York: Springer-Verlag, 1976

[45] 汪成为，高文，王行仁. 灵境 (虚拟现实) 技术的理论、实现与应用. 北京：清华大学出版社，1996

[46] 王铎主编. 理论力学习题选集. 北京：人民教育出版社，1963

[47] 王方良. 自动控制原理. 北京：科学出版社，2001

[48] 王永生，刘静华，杨光红. Turbo Pascal 实用绘图程序与图形. 北京：北京航空航天大学出版社，1994

[49] 武汉大学，山东大学. 计算方法. 北京：人民教育出版社，1979

[50] Xue Dingyü, Analysis and computer-aided design of nonlinear systems with Gaussian inputs. D.Phil. thesis, the University of Sussex, 1992

[51] 薛定宇. 控制系统计算机辅助设计 — MATLAB 语言及应用. 北京：清华大学出版社，1996

[52] 薛定宇. LaTeX 科学文件处理软件入门 (修订版). 东北大学控制仿真中心内部讲义，1997

[53] 薛定宇. 反馈控制系统的分析与设计 — MATLAB 语言应用. 北京：清华大学出版社，2000

[54] 薛定宇. 科学运算语言 MATLAB 5.3 程序设计与应用. 北京：清华大学出版社，2000

[55] Xue D, and Atherton D P, Simulation analysis of continuous systems driven by Gaussian white noise, in Jamshidi M, and Herget C J (eds.), Recent advances in computer-aided control systems engineering, Elsevier Science Publisers B V, 431-452, 1992

[56] 严子翔. VRML 虚拟现实网页语言. 北京：清华大学出版社，2001

索 引